CLUSTER DE ALTA DISPONIBILIDADE ATRAVÉS DE ESPELHAMENTO DE DADOS EM MÁQUINAS REMOTAS

DANIEL ZAMINHANI

2008

DANIEL ZAMINHANI

***CLUSTER* DE ALTA DISPONIBILIDADE ATRAVÉS DE ESPELHAMENTO DE DADOS EM MÁQUINAS REMOTAS**

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Marluce Rodrigues Pereira

LAVRAS
MINAS GERAIS – BRASIL
2008

DANIEL ZAMINHANI

CLUSTER DE ALTA DISPONIBILIDADE ATRAVÉS DE ESPELHAMENTO DE DADOS EM MÁQUINAS REMOTAS

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em _____ de ___________________ de _____

____
Prof. Joaquim Quinteiro Uchôa

____
Prof. Sandro Pereira Melo

____
Prof. Marluce Rodrigues Pereira
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

A minha querida esposa, pelo apoio e compreensão, e aos meus pais, pela orientação que me deram ao longo da minha vida, ensinando-me a respeitar o próximo e crescer como homem.

Resumo

A indisponibilidade de um servidor que hospeda um site de *e-commerce* ou o serviço de correio eletrônico de uma empresa reflete negativamente no relacionamento da empresa com seus clientes e fornecedores, conseqüentemente impactando no seu faturamento. O objetivo deste trabalho é apresentar uma solução, utilizando *software* livre, para a criação de um *cluster* de alta disponibilidade através do espelhamento de dados em máquinas remotas para hospedar uma aplicação *SMTP*. Os testes realizados para validação da solução mostraram que o *cluster* montado garante alta disponibilidade do serviço *SMTP*.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

APT	*Advanced Packaging Tool*
CPU	*Central Processing Unit*
CRC	*Cyclic Redundancy Check*
DNS	*Domain Name System*
DRBD	*Distributed Replicated Block Device*
FQDN	*Fully Qualified Domain Name*
GFS	*Global File System*
GPL	*General Public Licence*
GNU	*GNU is Not Unix*
HA	*High Availability*
HTTP	*Hypertext Transfer Protocol*
I/O	*Input/Output*
IP	*Internet Protocol*
MD5	*Message-Digest Algorithm 5*
MPI	*Message Passing Interface*
MTA	*Message Transfer Agent*
NASA	*National Aeronautics and Space Administration*
NBD	*Network Block Device*
NFS	*Network File System*
OCFS2	*Oracle Cluster File System*
PVM	*Paralell Virtual Machine*
RAID	*Redundant Array of Independent Disks*
RAM	*Random Access Memory*
SAN	*Storage Area Network*

SCSI	*Small Computer System Interface*
SHA1	*Secure Hash Algorithm 1*
SMTP	*Simple Message Transfer Protocol*
SPOF	*Single Point Of Failure*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

1. INTRODUÇÃO

O grande avanço tecnológico tornou a dependência dos sistemas computacionais cada vez maior na vida das pessoas e no dia-a-dia das empresas. Transações bancárias, compra e venda de produtos, comunicação interpessoal e concretização de negócios são realizados eletronicamente a todo instante. Os recursos tecnológicos deixaram de ser apenas uma ferramenta de trabalho, tornando-se um fator competitivo de suma importância para pequenas, médias e grandes empresas.

A indisponibilidade de um servidor que hospeda um site de *e-commerce* ou o serviço de correio eletrônico, reflete negativamente não apenas no faturamento de uma empresa, mas também no relacionamento com seus clientes e fornecedores. Atualmente, o *site* de uma empresa é o seu cartão de visitas e o correio eletrônico um canal muito importante para comunicação entre entidades internas e externas à companhia.

Toda essa dependência torna crítico qualquer processo que envolva recursos computacionais, resultando em uma necessidade cada vez mais constante da disponibilidade dos serviços providos por esses recursos. O conceito de disponibilidade pode ser definido como o percentual de tempo em que o sistema e a aplicação estão operando, permitindo a livre utilização pelos usuários (TAURION, 2007) .

Para garantir a alta disponibilidade de um sistema computacional, além de um conjunto de *software* estável, é necessário também que exista redundância de *hardware* a fim de diminuir o *SPOF* (*Single Point Of Failure*) e conseqüentemente a probabilidade de interrupções não programadas. O objetivo

deste trabalho é apresentar uma solução que garanta a alta disponibilidade de um sistema computacional através da criação de um *cluster* utilizando o sistema operacional GNU/Linux. O *cluster* será criado para garantir a alta disponibilidade de um *gateway SMTP*, porém, os conceitos aqui apresentados podem ser utilizados em qualquer outra aplicação para o qual se deseja aplicar a alta disponibilidade.

1.1 MOTIVAÇÃO PARA REALIZAR ESTE TRABALHO

A realização deste trabalho foi motivada pela necessidade cada vez maior de manter o *uptime* do serviço *SMTP* em uma empresa, que utiliza este serviço tanto para o tráfego de *e-mail* com a internet como também para envio de alertas disparados pelos sistemas legados da companhia e pelos sistemas que monitoram o ambiente computacional. Essa criticidade proporcionou o desafio de buscar uma alternativa para manter a alta disponibilidade do serviço *SMTP* utilizando *software* livre e recursos de *hardware* existentes na empresa.

1.2 OBJETIVOS

O objetivo geral deste trabalho é resolver o problema do fornecimento, com alta disponibilidade, do serviço *SMTP* em uma empresa.

Devido à restrição de recursos computacionais, os objetivos específicos

deste trabalho compreendem alcançar uma solução utilizando recursos já existentes na empresa e *software* livre.

A solução encontrada para alcançar estes objetivos, foi criar um *cluster* de alta disponibilidade sobre o qual será executado o serviço *SMTP*. O *cluster* é composto por duas máquinas interligadas em rede, denominadas *host1* e *host2*. Cada *host* do *cluster* poderá estar em um dos seguintes estados: primário ou secundário. A aplicação é executada no *host* que estiver no estado primário e todos os dados são espelhados para o *host* secundário através de uma interface de rede dedicada. Para promover o *host* em estado secundário para primário, em caso de falha, será utilizado o *Heartbeat*. Quando a comunicação com o *host* que falhou for retomada, este passará então a ser o *host* secundário e, portanto, deverá sincronizar os dados com o *host* atualmente em estado primário.

O conjunto de *software* livre utilizado para prover a solução foi:

- Sistema operacional GNU/Linux.
- *Software DRBD* para sincronização dos dados entre os membros do *cluster*.
- *Software Heartbeat* para monitorar os nós e, em caso de falha, passar o controle de execução da aplicação para o outro *host*.
- Aplicação *Exim* para prover o serviço de *SMTP*.

Após a montagem e configuração do *cluster*, os resultados apresentados

mostraram que a solução foi eficiente, reduzindo a indisponibilidade do serviço e satisfazendo melhor as necessidades dos usuários.

1.3 ESTRUTURA DO TRABALHO

O texto está dividido em cinco capítulos incluindo esta introdução, que apresenta o problema e também os objetivos propostos.

O Capítulo 2 define o que é um *cluster* e quais são os tipos geralmente utilizados como: *cluster* de alto desempenho, *cluster* para balanceamento de carga e *cluster* de alta disponibilidade, que é o tema desta pesquisa e portanto terá uma abrangência maior no decorrer do trabalho.

O Capítulo 3 aborda o espelhamento de dados através de *RAID* e o espelhamento de dados em máquinas remotas, utilizando o *software DRBD*.

O Capítulo 4 apresenta a metodologia utilizada e os resultados alcançados. Serão discutidos os componentes que integram o *cluster*, incluindo o *hardware* e o sistema operacional utilizado, assim como o *software* escolhido para manutenção e gerenciamento do *cluster*. Também será apresentado o *Exim*, que é uma aplicação de *SMTP* que será executada sobre o *cluster* e os aspectos técnicos da solução, incluindo todo o processo de instalação e configuração desde o sistema operacional, passando pelo *software* de *cluster* até a aplicação. O Capítulo 4 também apresentará os testes realizados para validação da solução de *cluster*.

O Capítulo 5 apresenta conclusões e trabalhos futuros que podem ser desenvolvidos a partir dos conceitos abordados nesta pesquisa.

2 *CLUSTERS*

Este capítulo aborda a definição de cluster, suas vantagens e desvantagens, quando surgiram e como podem se classificados.

2.1 DEFINIÇÃO DE *CLUSTER*

De origem inglesa, a palavra *cluster* significa um conjunto ou grupo de coisas similares. Em computação, o termo *cluster* é utilizado para definir um grupo de dois ou mais computadores interligados, cujo objetivo é obter disponibilidade, desempenho ou aumentar de forma geral a capacidade de um sistema (JUNIOR E FREITAS, 2005).

Segundo (FERREIRA, 2003), *clusters* são definidos como “grupos de computadores configurados para trabalhar em conjunto com aplicações específicas” e ¨o modo como são configurados dá a impressão de serem um único computador”.

Para alcançar este objetivo, os membros do *cluster*, geralmente denominados de nós ou nodos, precisam se comunicar entre si para controlar todas as tarefas que devem ser executadas. (PEREIRA FILHO, 2004).

Do ponto de vista do usuário, o *cluster* é um sistema único independentemente da quantidade de nós que o compõe. Todos os aspectos relativos a distribuição de carga, dados, troca de mensagens, sincronização e organização física devem ser transparentes e abstraídos do usuário.

Dentre as vantagens oferecidas pelo *cluster* segundo (JUNIOR E

FREITAS, 2005), destacam-se:

- **Alto desempenho**

Possibilidade de resolver problemas complexos através de processamento paralelo, diminuindo assim o tempo gasto para execução das tarefas.

- **Escalabilidade**

Conforme aumenta a carga de trabalho, é possível aumentar também o poder de processamento através da adição de novos componentes ao *cluster*

- **Tolerância a falhas**

Permite a continuidade do serviço em caso de falhas, aumentando dessa forma a confiabilidade do sistema.

Os *clusters* podem ser aplicados em diferentes situações ou cenários e geralmente são utilizados nos casos onde é desejado manter a alta disponibilidade de serviços críticos, como páginas *web*, *e-mail* etc. Também utiliza-se *clusters* para obter alto desempenho em aplicações que requeiram grande poder de processamento. A Seção 2.2 detalha os diferentes tipos de *cluster* e onde são geralmente utilizados.

2.2 TIPOS DE *CLUSTER*

2.2.1 Alto desempenho

Como o próprio nome já diz, esse tipo de *cluster* tem foco na obtenção de desempenho para aplicações que precisam de grande poder de processamento na execução de atividades complexas. Uma grande tarefa computacional pode ser dividida em tarefas menores e então distribuída através dos nós de um *cluster*, denominando-se então de processamento distribuído ou processamento paralelo, reduzindo dessa forma o tempo gasto na execução total da tarefa. O processamento paralelo caracteriza-se pela distribuição dos processos para mais de uma *CPU* e, o processamento distribuído, caracteriza-se pelo compartilhamento dos recursos de computadores otimizando dessa forma uma tarefa computacional (ALVARENGA, 2007).

É comum encontrarmos nas fontes de pesquisa a associação de *clusters* ao projeto *Beowulf*[1] da *NASA*, principalmente em referência aos *clusters* de alto desempenho, por isso, muitos autores simplesmente denominam um *cluster* de alto desempenho como C*luster Beowulf*. Segundo (FERREIRA, 2003), "*Beowulf* é uma arquitetura multicomputador que pode ser usada para computações paralelas. É um sistema que normalmente consiste em um nó servidor e um ou mais nós clientes conectados via rede". Nenhum *hardware* especialmente feito sob encomenda é requerido, sendo necessário para criação de um *cluster Beowulf* apenas componentes de *hardware* comuns e o sistema operacional Linux, além das bibliotecas *PVM* ou *MPI*. É por esse motivo que o custo total de uma solução de *cluster Beowulf* é relativamente baixo se comparado ao custo de supercomputadores, sendo este, um grande atrativo para

1 http://www.nas.nasa.gov/SC2000/GSFC/beowulf.html

criação de *clusters* utilizando conjuntos de *hardware* comuns encontrados no mercado.

Os *clusters* de alto desempenho são utilizados geralmente em projetos científicos ou em análises financeiras, onde grandes cálculos matemáticos são necessários exigindo dessa forma um grande poder de processamento.

A Figura 2.1 ilustra um modelo de cluster de alto desempenho, composto por um nó controlador e 24 nós escravos interligados por um *switch*. A comunicação entre os nós ocorre por uma rede *Ethernet*.
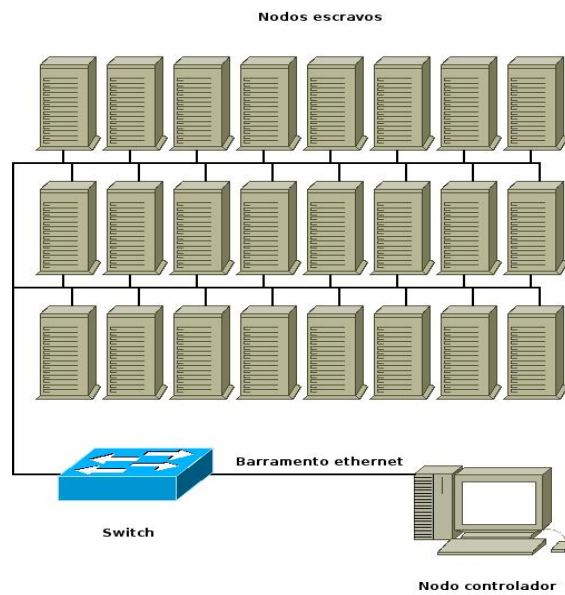
Figura 2.1: Modelo de cluster de alto desempenho ou Beowulf

2.2.2 Balanceamento de carga

No modelo de *Clusters* de balanceamento de carga, todos os nós executam as mesmas aplicações e o tráfego entrante é então dividido de forma equilibrada entre eles. Se houver falha em um algum, o as requisições serão então redistribuídas para os nós disponíveis no momento. O *cluster* de balanceamento de carga, apesar de ter como objetivo principal o aumento do desempenho, difere do modelo de *cluster* de alto desempenho pois as requisições dos clientes são atendidas completamente pelo nó ao qual a requisição foi direcionada. O objetivo neste caso é dividir as requisições e não as sub-tarefas nelas envolvidas (SIMÕES e TORRES, 2003). Este modelo de *cluster* é muito utilizado na internet para distribuir a carga de servidores de páginas e *e-mail*. A Figura 2.2 ilustra o funcionamento de um *cluster* para balanceamento de carga. Neste exemplo, existe 1 nó controlador, denominado de balanceador, que é responsável pelo recebimento das requisições e pela distribuição delas entre os 5 nós que compõe o *cluster*.
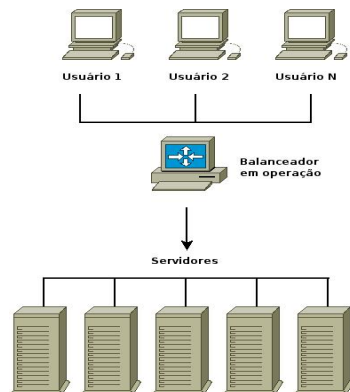
Figura 2.2: Modelo de cluster para balanceamento de carga

2.2.3 Alta disponibilidade

Segundo (FERREIRA, 2003), “alta disponibilidade é uma técnica que consiste na configuração de dois ou mais computadores para que eles passem a trabalhar em conjunto. Dessa forma, cada computador monitora os demais e, em caso de falhas, assume os serviços que ficaram indisponíveis”. A esse tipo de *cluster*, da-se o nome de *cluster* de alta disponibilidade, ou *HA*, do inglês *High Availability*, sendo este modelo o foco desse trabalho.

Clusters de alta disponibilidade são usados quando existe a necessidade de fornecer serviços que estejam sempre, ou quase sempre, disponíveis para receber requisições. Este nível de serviço é um fator dependente do *cluster* (PEREIRA FILHO, 2004).

A disponibilidade pode ser classificada em três classes distintas (FERREIRA, 2003):

- **Disponibilidade convencional**

É a disponibilidade encontrada na maioria dos computadores disponíveis no mercado, sem nenhum recurso extra que de alguma forma oculte eventuais falhas. Os recursos computacionais pertencentes a essa classe possuem uma disponibilidade entre 99% a 99,9%, ou seja, em um ano de operação poderá existir indisponibilidade por um período de nove horas a quatro dias.

- **Alta disponibilidade**

É encontrada em computadores que possuem algum recurso para detectar, recuperar e ocultar eventuais falhas. Os recursos computacionais pertencentes a essa classe possuem uma disponibilidade entre 99,99% a 99,999%, ou seja, em um ano de operação poderá existir indisponibilidade por um período de pouco mais de cinco minutos.

- **Disponibilidade contínua**

É encontrada em computadores sofisticados que possuem recursos muito eficientes para detectar, recuperar e ocultar eventuais falhas obtendo-se disponibilidade cada vez mais próxima de 100%, tornando dessa forma o período de inatividade insignificante ou até mesmo inexistente.

É comum encontrar o termo *tolerância a falhas* para indicar sistemas de alta disponibilidade. Segundo Jalot, apud (PEREIRA FILHO, 2004), um sistema é tolerante a falhas se ele pode mascarar a presença de falhas. Gartner (2007) demonstra que para tolerar falhas é preciso ter redundância e, que em sistemas, redundância pode ser *hardware*, *software* ou tempo. Dessa forma, compreende-se que a redundância de recursos é um pré-requisito para alcançar alta disponibilidade em um sistema computacional, portanto, a ausência de redundância acarretará em *pontos únicos de falha – SPOF*.

Segundo Marcus e Stern, apud (PEREIRA FILHO, 2004), muitos projetos de alta disponibilidade são ineficientes porque não contemplam a contingência de todos os recursos necessários para disponibilidade do sistema, como equipamentos de rede e energia elétrica por exemplo, permitindo dessa forma a presença de *SPOF* no ambiente.

A Figura 2.3 ilustra um modelo de *cluster* de alta disponibilidade

utilizando um sistema de armazenamento de dados compartilhado, conectado ao *cluster* via conexões redundantes de fibra óptica.
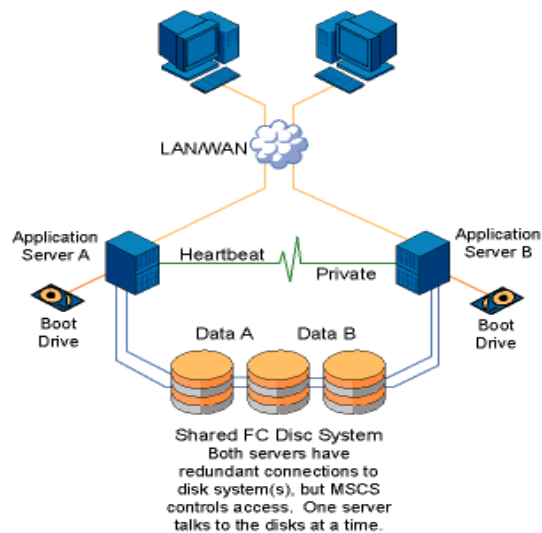
Figura 2.3: Modelo de cluster de alta de disponibilidade. Fonte: (Dell, 2008)

Conforme informado anteriormente, o objetivo deste trabalho é criar um *cluster* de alta disponibilidade para executar uma aplicação *SMTP*. Uma solução possível para alcançar esse objetivo é baseada em quatro elementos: o sistema operacional GNU/Linux, um sistema de arquivos robusto, um *software* para espelhamento de dados e um *software* para monitoramento dos nós do *cluster*. Os procedimentos para implementar essa solução serão detalhados no Capítulo 4.

2.2.4 Discussão

Cada tipo de *cluster* é melhor utilizado na solução de um determinado tipo de problema. Para resolver o problema de alta disponibilidade que está sendo abordado neste trabalho, o *cluster* de alta disponibilidade é a solução mais apropriada.

Para construir um *cluster* de alta disponibilidade é necessário definir a forma como os dados serão armazenados de forma a manter a alta disponibilidade. No Capítulo 3 serão abordados os vários tipos de espelhamento que podem ser utilizados em um *cluster* de alta disponibilidade.

3 ESPELHAMENTO DE DADOS

Este capítulo apresenta os diferentes tipos de espelhamento de dados existentes e que podem ser utilizados em um *cluster*. Além disso, são apresentados os algoritmos necessários para realizar o espelhamento de dados em máquinas remotas.

3.1 TIPOS DE ESPELHAMENTO DE DADOS

3.1.1 Espelhamento de discos

Em sistemas de missão crítica, redundância é um termo que deve ser levado a sério para que seja possível garantir a continuidade dos serviços fornecidos por um sistema computacional. Uma forma de aumentar a disponibilidade e diminuir o *SPOF*, é tentar garantir que um dado armazenado em disco esteja disponível mesmo quando esse disco apresentar falha. Isso é possível quando o dado além de ser gravado em um disco seja também copiado (espelhado) para outro, criando-se portanto a redundância. Neste caso, será necessário dois ou mais dispositivos, caracterizando-se um *array* de discos.

Os *arrays* de discos são conhecidos como *RAID* (*Redundant Array os Independent Disks*) e são usados, atualmente, para melhorar o desempenho e a confiabilidade de um sistema de armazenamento armazenando dados redundantes (SILBERSCHATZ, 2004).

Existem diferentes possibilidades de configuração do *array*, permitindo

a criação de *arrays* dedicados à redundância de dados, aumento de desempenho, expansão da capacidade de armazenamento ou uma combinação dessas características. O *RAID* utiliza diferentes técnicas para armazenar e recuperar dados, como: *striping*, espelhamento e paridade. A implementação de *RAID* pode ocorrer através de *software* ou *hardware*, sendo o custo e o desempenho significativamente inferiores no primeiro caso. Também existem alguns níveis de *RAID*, cada qual com seu próprio método para manipular os dados que são armazenados em disco. A escolha do melhor nível dependerá da necessidade da aplicação e do que deve ser priorizado.

- ***RAID* 0**

O *RAID* nível 0, ilustrado na Figura 3.1, também conhecido como *Striping*, consiste na divisão dos dados em segmentos de igual tamanho, que são dessa forma gravados nos discos do *array* proporcionando melhor desempenho tanto no processo de leitura como gravação. O *RAID* 0 não provê a redundância dos dados, portanto, se um disco falhar, todo o volume lógico falhará.
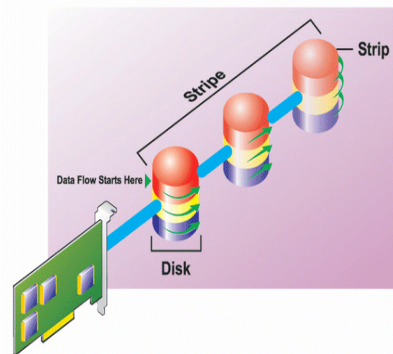
Figura 3.1: RAID 0 – Divisão dos dados. Fonte: (Dell, 2008)

- **RAID 1**

O *RAID* nível 1, ilustrado na Figura 3.2, é a maneira mais simples para manter a redundância dos dados. Neste caso, os dados gravados em um disco são espelhados (duplicados) em outro, portanto, se houver uma falha, o *array* poderá ser reconstruído a partir do disco íntegro. O *RAID* 1 provê melhor desempenho no processo de leitura, porém, é mais lento no processo de gravação devido a necessidade de duplicação dos dados para prover redundância.
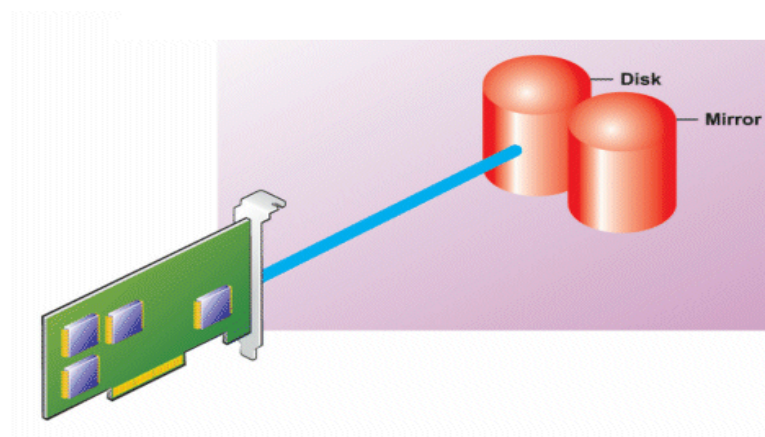
Figura 3.2: RAID 1 - Espelhamento de dados. Fonte: (Dell, 2008)

- **RAID 5**

O *RAID* nível 5, ilustrado na Figura 3.3, consiste na divisão dos dados, assim como ocorre no *RAID* nível 0, associado com a distribuição da paridade dos dados alternadamente entre os discos do *array*. A vantagem em relação ao

RAID 1 é que neste caso não é necessário um par para cada disco utilizado, sendo o tamanho lógico do *array* igual a capacidade de n-1 disco, o que provê além da redundância um bom aproveitamento da capacidade de armazenamento.
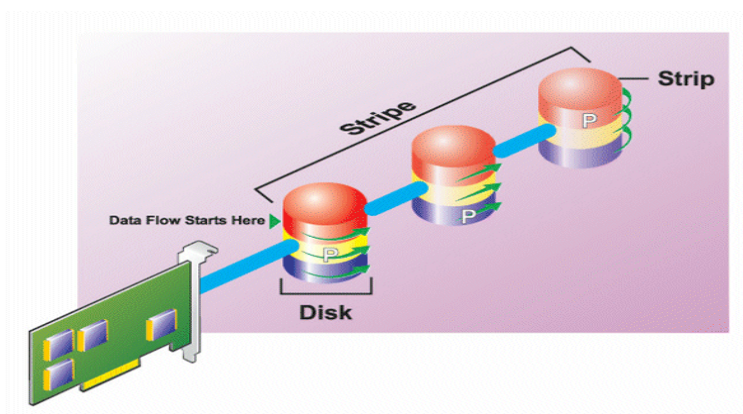
Figura 3.3: RAID 5. Fonte: (Dell, 2008)

É possível também obter níveis de *RAID* que derivam da combinação dos níveis aqui apresentados, como por exemplo: *RAID* 10, *RAID* 0+1 e *RAID* 50. Além disso, existem os níveis 2, 3, 4 e 6, descritos em (SILBERSCHATZ, 2004), cuja utilização é menos comum.

O *RAID* pode ser implementado através de *hardware*, utilizando uma placa controladora que gerencia os níveis de RAID e processa as operações de I/O nos *arrays*, ou através de software, utilizando funções do sistema operacional, porém, com desempenho muito inferior a qualquer implementação de *RAID* via *hardware*.

Devido às características de desempenho e redundância encontradas em cada nível de *RAID*, conclui-se que o método mais adequado dependerá da aplicação envolvida. A tabela 3.1 resume essas características e serve como base de comparação entre elas.

Tabela 3.1: Comparação de desempenho entre os níveis de RAID. Fonte: (Dell, 2008)

Nível de RAID	**Disponibilidade do dado**	**Desempenho de leitura**	**Desempenho de gravação**	**Desempenho na reconstrução dos dados**	**Quantidade mínima de discos**	**Recomendação de uso**
0	Baixa	Muito bom	Muito bom	-	N	Dados não críticos
1	Excelente	Muito bom	Bom	Bom	2 (N=1)	Pequenas bases de dados, *logs*, dados críticos
5	Boa	Leitura seqüencial: Boa/Leitura transacional: Muito boa	Razoável	Ruim	N+1 (N>=2)	Bases de dados e aplicações com alta intensidade de leitura
10	Excelente	Muito bom	Razoável	Bom	2N x Y	Aplicações com grande volume de dados
50	Excelente	Muito bom	Razoável	Razoável	N+2 (N>=4)	Aplicações com grande volume de dados
N = Número de discos no *array* Y = Número de conjuntos *RAID*						

3.1.2 Espelhamento de dados em máquinas remotas

O espelhamento de dados em máquinas remotas é uma solução eficiente quando o objetivo é criar um ambiente de alta disponibilidade para serviços

críticos , isso porque, é possível obter a redundância não apenas dos dados, como no caso do *RAID*, mas de todo o conjunto de *hardware* existente*,* diminuindo significativamente o *SPOF*.

> "*Em um ambiente de espelhamento de discos, não há nenhum tipo de compartilhamento entre os nós servidores. Através do uso de software, os dados são espelhados ou replicados de um servidor para o outro através da rede. O princípio deste modelo é que todos servidores potenciais substitutos devem possuir seu próprio meio de armazenamento e, ao mesmo tempo, uma réplica do armazenamento do servidor a ser substituído*" (SIMÕES e TORRES, 2003).

Alguns autores denominam essa técnica como sendo um *RAID* 1 via TCP/IP, ou seja, um espelhamento de disco através da rede. Todo dado gravado em disco é duplicado em outro, porém, em máquinas diferentes, garantindo que se houver qualquer falha de *hardware*, a máquina de *backup* esteja pronta para a operação de *failover*, assumindo imediatamente os serviços. Esse processo pode ser manual ou automático, assim como o *failback*, ou seja, o retorno do serviço para a sua máquina de origem após a resolução da falha.

O *software* utilizado para fazer o espelhamento de disco em máquinas remotas será o *DRBD*, cujas características serão detalhadas no Capítulo 4.

4 METODOLOGIA E RESULTADOS

Neste capítulo serão abordados os componentes que integram o *cluster*, incluindo o *hardware* e o sistema operacional utilizado, assim como o *software* escolhido para manutenção e gerenciamento do *cluster*. Os aspectos técnicos da solução, como instalação e configuração dos componentes, também serão abordados.

4.1 COMPONENTES DO *CLUSTER*

4.1.1 Hardware

Duas máquinas foram utilizadas para a criação do *cluster*. A configuração mínima de cada máquina dependerá dos requisitos exigidos pela aplicação que será hospedada. Para um serviço de *SMTP* com carga média de 35000 mensagens/dia, foram utilizadas máquinas com a seguinte configuração cada:

- ✔ Processador .... 2 Pentium III de 500MHz
- ✔ Memória .... 768 MB
- ✔ Disco .... 2 discos *SCSI* de 9GB
- ✔ Rede .... 2 placas de 100 Mbit

É importante destacar que as máquinas que irão compor o *cluster* não precisam necessariamente ter as mesmas configurações de *hardware*, no entanto, um pré-requisito importante é a existência de espaço em disco suficiente para a criação de uma partição de mesmo tamanho em cada uma delas.

4.1.2 Sistema operacional

Uma das condições para realização desse trabalho, é o uso integral de *software* livre em todos os componentes do *cluster*. Dessa forma, o sistema operacional utilizado foi o GNU/Linux, que além de ser *software* livre, oferece segurança, desempenho e confiabilidade.

A distribuição escolhida foi o *Debian* que, entre outras características, possui uma filosofia que se aproxima ao máximo das diretrizes do movimento *Software* Livre e utiliza, por padrão, apenas *software* distribuído sob licença de código aberto. O Debian também possui um sistema de pacotes próprio e oferece ferramentas como o *APT* e o *Aptitude*, que tornam o processo de instalação de aplicativos muito simples e eficiente, principalmente quanto à satisfação de dependências. Mais informações sobre o *Debian* assim como orientações sobre como adquirir uma cópia do sistema, podem ser obtidas no site http://www.debian.org.

4.1.3 DRBD

O *DRBD*[2], *Distributed Replicated Block Device*, desenvolvido por Philipp Reisner e Lars Ellenberg, é um dispositivo de bloco para sistemas Linux, implementado através de um módulo do *kernel*, que permite o espelhamento em tempo real de um dispositivo de bloco local para uma máquina remota.

A função do *DRBD* é manter a disponibilidade dos dados e conseqüentemente a continuidade dos negócios que dependem de um sistema computacional. Essa disponibilidade é uma questão muito séria e pode se tornar um grande problema para um projeto de *cluster*. Se os dados devem estar disponíveis a maior parte do tempo possível, então um sistema de armazenamento compartilhado como *SAN*[3], *NFS*[4] *e NBD*[5], por exemplo, podem não ser a melhor escolha, pois tornam-se um ponto de falha não redundante além de ser uma solução de alto custo financeiro no caso da *SAN*.

Outros métodos de espelhamento de disco, como o *RAID*, explicado no Capítulo 3, ou o *rsync,* tornam-se ineficientes em uma solução de alta disponibilidade. O *RAID* resolverá o problema apenas quando houver falha de disco, porém, não haverá redundância para os demais itens de *hardware*, o que pode gerar a indisponibilidade em caso de falha. A sincronização remota de dados via *rsync* não é uma alternativa adequada para uma solução de alta disponibilidade porque, no caso de falha do nó principal do *cluster*, o secundário assumirá os serviços com perda dos dados alterados desde a última sincronização.

Este tipo de problema é superado com o *DRBD*, uma vez que os dados são replicados entre os nós do *cluster* em tempo real. O dispositivo de bloco

---

2 http://www.drbd.org
3 http://www.gta.ufrj.br/grad/04_1/san
4 http://nfs.sourceforge.net
5 http://nbd.sourceforge.net

disponibilizado pelo *DRBD*, geralmente */dev/drbd0*, deverá estar presente em todos os nós do *cluster*, dessa forma, toda operação local de escrita em disco será espelhada no dispositivo de bloco remoto através da rede. Esta operação pode ser vista como um *RAID-1* via *TCP/IP*. Através da Figura 4.1, é possível observar que o *DRBD* é adicionado no processo de *I/O* antes do *driver* de disco efetivar o acesso, permitindo dessa forma que os dados sejam espelhados para o outro nó do *cluster* através da rede.

Todos os dispositivos disponibilizados pelo *DRBD* possuem um estado, que pode ser ***primário*** ou ***secundário***. A aplicação que estiver em execução no *cluster* deve obrigatoriamente ser executada no nó com estado ***primário*** e, somente este, terá permissão para acessar o dispositivo de bloco. Toda operação de escrita será realizada localmente e enviada para o nó que possuir o dispositivo em estado ***secundário*** que, por sua vez, irá simplesmente gravar os dados no seu dispositivo de bloco local. As operações de leitura ocorrerão somente no nó em estado ***primário***.

Se o nó ***primário*** do *cluster* falhar, o *Heartbeat* dará início ao processo de *failover* alterando automaticamente o estado do nó ***secundário*** para ***primário***, montando o dispositivo de bloco espelhado e iniciando a aplicação. Quando o nó que falhou for reativado, este receberá o estado ***secundário*** e deverá sincronizar o seu conteúdo com o nó promovido para ***primário***. A partir da versão 7, o *DRBD* é capaz de identificar apenas os dados que foram alterados durante a indisponibilidade do nó problemático, tornando a sincronização seletiva e conseqüentemente mais rápida.
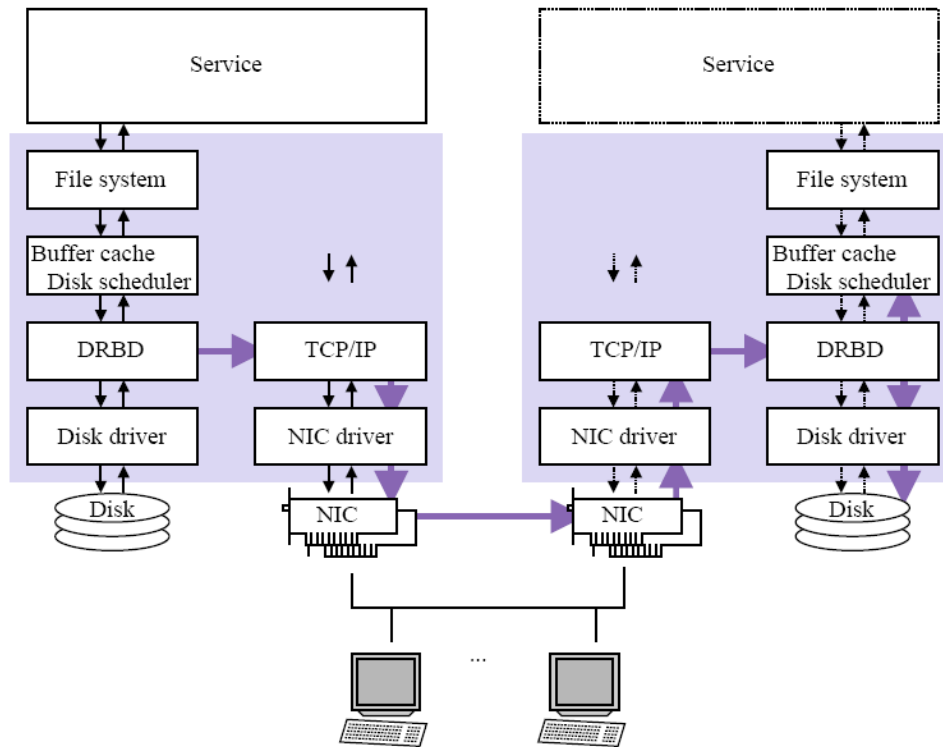
Figura 4.1: Funcionamento do DRBD (REISNER, 2001)

Segundo (REISNER, 2001), o *DRBD* possui 3 protocolos de operação que podem ser utilizados de acordo com a necessidade de implementação. São eles: *Protocolo A*, *Protocolo B* e *Protocolo C*. O usuário deve especificar o tipo de protocolo que deseja utilizar através do arquivo de configuração do *DRBD*, abordado na *seção 4.2.4* deste capítulo.

- **Protocolo A**

Utiliza um modo de operação assíncrono, ou seja, o *DRBD* assume como concluída a operação de espelhamento assim que os dados são gravados localmente e enviados para a rede. Neste caso, não existe a garantia de que os dados chegaram e foram gravados no nó secundário. Esta pode não ser a melhor configuração dependendo do serviço disponibilizado pelo *cluster*, porém, é com certeza a melhor opção quando o quesito desempenho é o mais importante.

- **Protocolo B**

Utiliza um modo de operação semi-síncrono, ou seja, o *DRBD* assume como concluída a operação de espelhamento somente quando receber a confirmação que o nó secundário do *cluster* recebeu o bloco de dados, no entanto, não existe a confirmação de que esses dados foram gravados no disco. Neste caso, é importante destacar que o *Protocolo B* oferece mais segurança do que o *Protocolo A*, no entanto, ainda existe um risco residual que pode não ser desejado dependendo da importância dos dados manipulados e do tipo de aplicação utilizada.

- **Protocolo C**

Utiliza um modo de operação síncrono, ou seja, o *DRBD* assume como

concluída a operação de espelhamento somente quando receber a confirmação que o nó secundário recebeu e gravou em disco o bloco de dados enviado. Este é o modo que requer um maior número de operações para concluir a tarefa, e também o de menor desempenho, no entanto, é o protocolo que oferece o nível mais alto de confiabilidade em todo o processo de espelhamento.

Na eventualidade de ter que decidir qual nó do *cluster* possui a cópia mais atual dos dados, o *DRBD* precisa manter uma área denominada de meta-dados. Essa área deve ser armazenada em um espaço não volátil em cada nó do *cluster* e pode ser *interna*, ou seja, no mesmo dispositivo onde são gravados os dados de produção, ou *externa*, armazenada em uma partição dedicada. Até a versão 7 o *DRBD* exige 128MB para a área de meta-dados, independente do tamanho do dispositivo de blocos que estiver em uso. A partir da versão 8, houve uma otimização do algoritmo permitindo definir o tamanho da área de meta-dados de acordo com o tamanho do dispositivo de bloco gerenciado pelo *DRBD*, conforme a tabela 4.1:

Tabela 4.1: Tamanho da área de meta-dados do DRBD. Fonte: (ELLENBERG, 2003)

Tamanho do dispositivo de bloco	Tamanho da área de meta-dados
1 GB	2 MB
100 GB	5 MB
1 TB	33 MB
4 TB	128 MB

Segundo (REISNER, 2001), os meta-dados são constituídos pelo ***inconsistent-flag*** e o ***generation-counter.*** O primeiro indica se o nó é alvo de um processo de sincronização ativo, ou seja, se está recebendo blocos de dados do nó primário naquele momento e portanto possui inconsistência. Já o *generation-counter* se divide em quatro partes:

- ✔ ***human-intervention-count***: é um contador que aumenta quando o estado do *cluster* é alterado por intervenção humana. Este contador possui a maior prioridade sobre todos os outros do *generation-counter.*

- ✔ ***connected-count***: é um contador que aumenta quando o estado do *cluster* muda e o nó é membro de um *cluster DRBD*, ou quando o nó secundário for desconectado do *cluster*.

- ✔ ***arbitrary-count***: é um contador que aumenta quando o estado do nó é alterado pelo *software* de gerenciamento do *cluster,* porém, aquele nó não faz parte de *cluster* em execução.

- ✔ ***primary-indicator***: 1 indica o nó primário em um *cluster DRBD* enquanto 0 indica o nó secundário.

Segundo (REISNER, 2001), quando existe comunicação entre os nós do *cluster*, o *generation-count* do nó primário sobrepõe o *generation-count* do nó

secundário, com exceção do *primary-indicator*. Durante a inicialização do *cluster*, é necessário que cada nó procure e se comunique com o seu par. Se isso não for possível, ele deve esperar até que a comunicação seja restabelecida ou deve se tornar o nó ativo através de intervenção humana.

Para identificar o nó que possui os dados mais atualizados, os componentes dos meta-dados são analisados na ordem mencionada acima. Se um nó com o *inconsistent-flag* ativado estiver presente, então o outro nó possui os dados mais atualizados. Se o *human-intervention-counters* for diferente entre os nós, aquele que tiver o maior valor possui os dados mais atualizados. O *connected-count* será utilizado somente se o *human-intervention* possuir o mesmo valor em todos os nós do *cluster*. Se o *connected-count* possuir o mesmo valor em todos os nós, então o *arbitrary-count* será utilizado e assim por diante.

Para ajudar o leitor a entender melhor como os meta-dados são utilizados pelo *DRBD*, serão apresentados alguns exemplos que simulam diversas situações e as decisões que são tomadas em cada um dos casos. Em cada tabela de exemplo, a primeira coluna apresenta o *generation-counter* do *host1*, a segunda coluna o *generation-counter* do *host2* e a terceira coluna descreve a situação. Os números separados por vírgula e envoltos entre o caractere **“<”** e o caractere **“>”**, significam o valor do *human-intervention-count*, do *connected-count*, do *arbitrary-count* e do *primary-indicator*, nesta ordem. O caracteres que precedem os valores do *generation-counter* indicam o estado do *host* no *cluster*, onde: ***“P”*** indica o *host* primário, ***“S”*** indica o *host* secundário, ***“-”*** indica que o *host* está inativo e ***“?”*** indica que o *host* possui estado indefinido no *cluster*.

Exemplo1: A tabela 4.2 apresenta a situação onde o *Host1* é o principal nó do *cluster* e portanto possui os dados mais atualizados. Dessa forma, ele deverá tornar-se o primário após uma reinicialização do *cluster*.

Tabela 4.2: Comportamento do DRBD – Exemplo1. Fonte: (REISNER, 2001)

Host1	Host2	Situação
P<0,0,0,1>	S<0,0,0,0>	Os dois nós estão no ar. O *generation-count* é igual.
P<0,1,0,1>	-<0,0,0,0>	O *Host2* cai e o *Host1* aumenta o *connected-count*
-<0,1,0,1>	-<0,0,0,0>	Os dois nós caem.
?<0,1,0,1>	?<0,0,0,0>	O cluster é reiniciado.
P<0,2,0,1>	S<0,2,0,0>	*Host1* atualiza o *generation-count* do *Host2* e o *connected-count* aumenta porque os nós estão novamente conectados.

Exemplo2: Este exemplo demonstra o comportamento do *DRBD* em uma situação típica em *clusters* de alta disponibilidade, ou seja, quando o *Host2* precisa assumir o papel de nó principal enquanto o *Host1* está indisponível.

Tabela 4.3: Comportamento do DRBD - Exemplo2. Fonte: (REISNER, 2001)

Host1	Host2	Situação
P<0,0,0,1>	S<0,0,0,0>	Os dois nós estão no ar. O *generation-count* é igual.
-<0,0,0,1>	S<0,0,0,0>	O *Host1* cai e o *Host2* não aumenta o *generation-count*.
-<0,0,0,1>	P<0,0,1,1>	O *Heartbeat* inicializa os serviços no *Host2*.
-<0,0,0,1>	-<0,0,1,1>	Os dois nós caem.
?<0,0,0,1>	?<0,0,1,1>	O *cluster* é reinicializado.
S<0,1,1,1>	P<0,1,1,1>	O *Host1* recebe a sincronização do *Host2* e o *Host2* se torna o primário.

Exemplo3: Este exemplo demonstra uma situação onde ocorre falta de energia e o *cluster* é reinicializado.

Tabela 4.4: Comportamento do DRBD - Exemplo3. Fonte: (REISNER, 2001)

Host1	Host2	Situação
P<0,0,0,1>	S<0,0,0,0>	Os dois nós estão no ar.
-<0,0,0,1>	-<0,0,0,0>	Ocorre o interrompimento de energia elétrica.
?<0,0,0,1>	?<0,0,0,0>	O *cluster* é reinicializado.
P<0,1,0,1>	S<0,1,0,0>	O *Host1*, que estava em estado primário antes da interrupção de energia, retorna em estado primário.

Exemplo4: Este exemplo demonstra o comportamento do *DRBD* e como os meta-dados são utilizados quando ocorre alguma intervenção humana.

Tabela 4.5: Comportamento do DRBD - Exemplo4. Fonte: (REISNER, 2001)

Host1	Host2	Situação
P<0,0,0,1>	S<0,0,0,0>	Os dois nós estão no ar.
-<0,0,0,1>	S<0,0,0,0>	O cabo de energia do *Host1* falha; O *Host2* não altera o *generation-count.*
-<0,0,0,1>	P<0,0,1,1>	O *Heartbeat* promove o *Host2* para primário.
-<0,0,0,1>	-<0,0,1,1>	Ocorre uma falha de disco no *Host2*.
?<0,0,0,1>	-<0,0,1,1>	O cabo de energia do *Host1* é reparado e o operador decide que é melhor inicializar o *Host1* imediatamente.
P<1,0,0,1>	?<0,0,1,1>	Misteriosamente o disco do *Host2* se recupera da falha, porém, a ordem do operador é aceita.
P<1,0,0,1>	S<1,0,0,0>	O *Host1* torna-se o primário.

Exemplo5: Neste último exemplo é demonstrado o comportamento do *DRBD* numa situação onde ocorre falha de rede.

Tabela 4.6: Comportamento do DRBD - Exemplo5. Fonte: (REISNER, 2001)

Host1	Host2	Situação
P<0,0,0,1>	S<0,0,0,0>	Os dois nós estão no ar e a rede está funcionando.
P<0,1,0,1>	S<0,0,0,0>	A rede para de funcionar. O *Host1* aumenta o *connected-count* pois entende que o *Host2* saiu do *cluster*.
P<0,1,0,1>	P<0,0,1,1>	O *Heartbeat* do *Host2* o promove para primário e inicializa os serviços, pois entende que o *Host1* está indisponível.
-<0,1,0,1>	-<0,0,1,1>	Ocorre interrupção de energia elétrica.
?<0,1,0,1>	?<0,0,1,1>	A energia e a rede são restabelecidas.
P<0,2,0,1>	S<0,2,0,0>	O Host1 retoma a posição de primário.

Além da operação normal onde os blocos de dados são espelhados, existem situações onde é necessário sincronizar todo o conteúdo do disco espelhado. O sincronismo foi projetado para não afetar a operação normal do nó que estiver executando a aplicação do *cluster* e ocorre paralelamente ao espelhamento normal. Para garantir que a aplicação do nó ativo não seja afetada pela sincronização, apenas uma parte da banda de rede é utilizada neste processo.

A forma utilizada para atualizar completamente o disco de um nó é a cópia de todos os blocos do nó primário para o secundário. Este método recebe o nome de ***sincronização total***. Se um nó é desconectado do *cluster* por um período curto de tempo, então será necessária apenas a cópia dos blocos que foram atualizadas durante o período de indisponibilidade. Este método recebe o nome de ***sincronização rápida*** (incremental).

A decisão mais importante que o *DRBD* precisa tomar é: Quando é necessária a sincronização e se ela deve ser total ou incremental. Para exemplificar, serão apresentados dois casos e a decisão tomada pelo *DRBD* sobre o tipo de sincronização que deve ser realizada em cada um deles.

Caso 1: Falha no nó secundário (REISNER, 2001)

Quando o nó secundário é desconectado do *cluster*, independente do que houver acontecido, isto não será um problema para o *DRBD*. Durante a indisponibilidade do nó secundário, todos os blocos gravados no nó primário serão marcados para permitir a sincronização incremental quando o nó secundário estiver novamente disponível.

Se o nó primário também falhar durante a indisponibilidade do nó secundário, então a tabela de marcação dos blocos pendentes será perdida, uma vez que esta reside na memória RAM. Dessa forma, da próxima vez que os dois nós estiverem disponíveis, será necessário efetuar a sincronização total.

Caso 2: Falha no nó primário (REISNER, 2001)

Se o nó primário falhar e retornar antes do nó secundário tornar-se o principal, então será necessária uma sincronização total do nó primário para o secundário. Isso ocorre porque não é possível determinar quais blocos gravados pouco antes da falha alcançaram o nó secundário, e a sincronização total é portanto a única forma de garantir a integridade dos dados.

Se o nó secundário tornar-se o principal durante a falha do nó primário, então a sincronização também será total, desta vez no sentido inverso. Isto é necessário pelo mesmo motivo da situação anterior.

Se os dois nós ficam indisponíveis, por interrupção de energia elétrica por exemplo, então ocorrerá sincronização total do nó em estado primário para o

nó em estado secundário logo após a reinicialização do *cluster*.

É importante ressaltar que o estilo de operação do *DRBD* discutido até aqui permite a execução da aplicação em apenas um dos nós por vez, enquanto o segundo nó permanece pronto para assumir as atividades do principal em caso de falha (Ativo/Passivo). Isto pode ser visto como um desperdício de recurso, uma vez que o nó secundário fica ocioso a maior parte do tempo. Segundo (ELLENBERG, 2007), a partir da versão 8 do *DRBD* é possível que, dependendo da aplicação, a execução ocorra em todos os nós do *cluster* simultaneamente (Ativo/Ativo). Para tornar isso possível é necessária a utilização de um sistema de arquivos exclusivo para *cluster*, como o *OCFS2*[6] e o *GFS*[7] por exemplo. Como a abordagem deste trabalho é *cluster* de alta disponibilidade, a utilização do DRBD no modo Ativo/Ativo não será discutida.

4.1.4 Heartbeat

O *Heartbeat*[8] é um software de código aberto, escrito por Alan Robertson, que tem a função de fazer o gerenciamento de *clusters* de alta disponibilidade. Por gerenciamento entende-se o monitoramento dos nós e tudo aquilo que deve ser feito para que o nó secundário assuma a aplicação do *cluster* quando houver indisponibilidade do nó primário.

O Heartbeat deve ser instalado e configurado em todos os nós e o seu

6 http://oss.oracle.com/projects/ocfs2
7 http://www.redhat.com/gfs
8 http://www.linux-ha.org/heartbeat

funcionamento "baseia-se no envio de sinais por parte dos membros do *cluster*. Com base nas respostas obtidas, os membros conseguem determinar se os outros integrantes estão ou não funcionando" (SIMÕES e TORRES, 2003). A comunicação pode ocorrer via placa de rede, conexão serial ou ambas. A Figura 4.2 ilustra o funcionamento do *Heartbeat* utilizando dois canais de comunicação entre os nós. A comunicação redundante visa diminuir pontos de falha que poderiam gerar uma falsa identificação de inatividade, onde os dois nós ficariam em estado primário simultaneamente. A comunicação ocorre via conexão serial e via interface de rede.
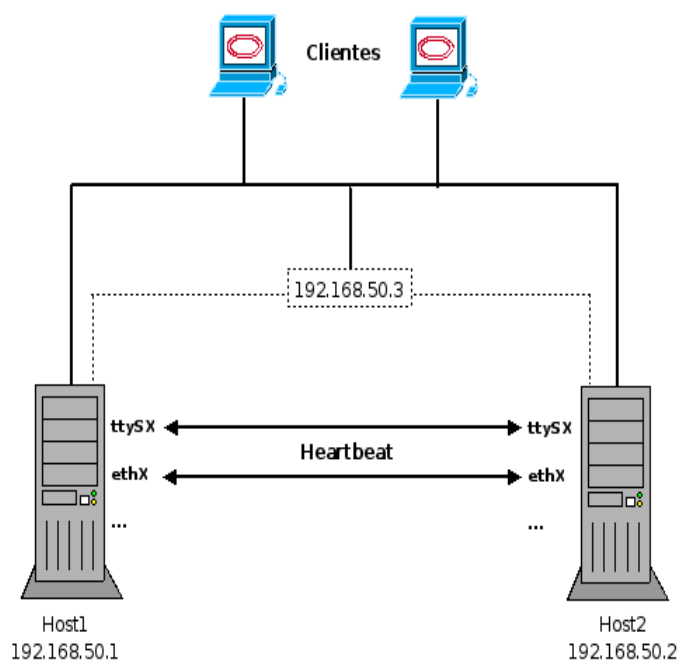
Figura 4.2: Visão geral do Heartbeat

As falhas são detectadas quando o *Heartbeat* deixa de receber sinais dos outros nós do *cluster*, por exemplo, se o *Heartbeat* do nó secundário deixa de receber os sinais enviados pelo *Heartbeat* do nó primário, então ele assume que existe uma falha e dá início ao processo de *failover*, ou seja, inicia os serviços necessários para que a aplicação seja executada no servidor *backup,* preservando assim o funcionamento do sistema como um todo. Essa operação abrange a promoção do nó secundário para primário, a montagem no modo leitura e escrita da partição que contém os dados replicados pelo *DRBD* e a configuração do *IP* virtual do *cluster* no nó secundário, que passa a responder pelas requisições feitas à aplicação.

Através do arquivo de configuração do *Heartbeat* é possível determinar se o processo de *failback*, ou seja, o retorno dos serviços para a máquina original após resolução do problema, deve ser automático ou não. Algumas vezes o nó onde a aplicação está sendo executada é indiferente, portanto, o processo automático de *failback* é indesejado.

O procedimento de instalação do *Heartbeat* será abordado na seção 4.2.5 deste capítulo, assim como os parâmetros de configuração disponíveis para criação do *cluster* em conjunto com o *DRBD*.

4.1.5 Aplicação

O *DRBD* em conjunto com o *Heartbeat* pode ser utilizado para a criação de *clusters* capazes de executar praticamente qualquer tipo de aplicação que necessite de alta disponibilidade, como por exemplo servidores *HTTP*, *SMTP*,

banco de dados, *DNS*, *firewall* etc. Neste trabalho, o *cluster* será criado para um serviço de *SMTP* utilizando o *Exim*.

O *Exim* é um agente para transporte de mensagens de correio eletrônico (*MTA*), similar ao *Sendmail*[9] e o *Postfix*[10], que é instalado por padrão no *Debian*. Desenvolvido na Universidade de Cambridge para uso em sistemas comuns ao Unix, como é o caso do GNU/Linux, o *Exim* é licenciado sob a *GPL* e oferece uma extensa variedade de configurações que o tornam muito flexível na tarefa de roteamento e checagem das mensagens assim como na integração com *software* externo.

9 http://www.sendmail.org
10 http://www.postfix.org

4.2 INSTALAÇÃO E CONFIGURAÇÃO DOS SERVIÇOS

4.2.1 Configuração do sistema operacional

O *Debian*[11] mantém três versões distintas que podem ser utilizadas por qualquer pessoa e que estão divididas da seguinte forma: ***estável***, ***teste*** e ***instável***.

A versão ***estável*** é a distribuição que sempre conterá a última versão oficial do sistema. Embora não contenha as últimas versões de *software* que compõe a distribuição, é a versão mais indicada para uso em produção pois garante a maior estabilidade possível utilizando somente versões maduras de *software*. A versão ***teste*** contém pacotes de *software* mais recentes, porém, que ainda não foram aceitos na versão ***estável***. O *software* é disponibilizado na versão ***teste*** somente depois que passa por um grau de testes na versão ***instável***. A versão ***instável*** é onde ocorre o desenvolvimento do *Debian* e portanto é a distribuição utilizada pelos desenvolvedores do sistema. As últimas versões de *software* estão presentes na versão ***instável*** do *Debian* e podem conter falhas justamente por estarem em fase de desenvolvimento.

Por se tratar de um ambiente de produção, será utilizada a versão ***estável*** do *Debian*, que até a data de conclusão deste trabalho é a 4.0r3 com a versão 2.6.18-4 do *kernel* do Linux. Os detalhes sobre o processo de instalação podem ser obtidos no "***Manual de Instalação Debian GNU/Linux***", disponível em http://www.debian.org/releases/stable/installmanual.

É muito importante que seja feita uma instalação básica do sistema e que

11 http://www.debian.org

sejam adicionados apenas o *software* necessário para execução do *cluster* e da aplicação. Essa medida pode proporcionar um maior desempenho e diminuir os riscos de possíveis vulnerabilidades existentes em *software* que não será utilizado.

O mesmo procedimento de instalação pode ser adotado nas duas máquinas que irão compor o *cluster*. Não é necessário que os dois equipamentos possuam a mesma configuração de *hardware.* No entanto, é imprescindível que exista espaço em disco suficiente para criação de uma partição do mesmo tamanho em cada um deles. Essa partição será utilizada para armazenar os dados *da aplicação,* que serão replicados pelo *DRBD* entre as máquinas.

Cada uma das máquinas utilizadas possui dois discos *SCSI* de 9GB. No primeiro disco foi instalado o sistema operacional e no segundo foi criado uma única partição para armazenar os dados da aplicação. A tabela 4.7 contém o esquema de particionamento utilizado:

Tabela 4.7: Esquema de particionamento dos discos

Partição	**Tamanho (MB)**	**Sistema de arquivos**	**Ponto de montagem**
/dev/sda1	8448	JFS	/
/dev/sda2	768	SWAP	-
/dev/sdb1	9216	JFS	-

A partição */dev/sdb1* será utilizada para o espelhamento e por isso não possui ponto de montagem. O dispositivo de bloco */dev/drbd0*, que será criado após a instalação do *DRBD*, será associado a partição */dev/sdb1* e utilizado na

montagem do dispositivo. Os detalhes desse procedimento serão abordados na seção 4.2.4 deste capítulo.

É muito importante que seja utilizado um sistema de arquivos com tecnologia *journaling*, caso contrário, será necessário verificar a consistência do sistema de arquivos com o *fsck* sempre que o nó secundário assumir os serviços do *cluster* após falha do nó primário.

O *fsck* consegue prover resultados satisfatórios, mas a correção de erros pode levar muito tempo, o que é inaceitável em um ambiente de alta disponibilidade. Além disso, se a falha no nó primário ocorreu quando dados estavam sendo gravados no disco, o *fsck* não conseguirá completar a gravação, resultando em perda de dados

Sistemas de arquivos com a tecnologia *journaling* possuem a capacidade de acompanhar as atualizações de dados que são feitas no sistema de arquivos antes que realmente sejam feitas. As informações que o *journaling* captura são armazenadas em uma área separada do sistema de arquivos, denominada *journal*, também conhecida como **registros de *log***.

Os registros de *log* são escritos antes que as mudanças efetivamente ocorram no sistema de arquivos e esses registros somente são eliminados quando as mudanças são feitas. Assim, se o computador é indevidamente desligado, o processo de montagem na próxima inicialização verificará se há mudanças gravadas no *journal* "marcadas" como não feitas. Se houver, tais mudanças serão aplicadas no sistema de arquivos, reduzindo o risco de perda de dados.

Existem vários sistemas de arquivos disponíveis com a tecnologia *journaling*, como o *XFS*, desenvolvido originalmente pela *Silicon Graphics*[12] e

12 http://www.sgi.com

posteriormente disponibilizado com código aberto, o *ReiserFS*, desenvolvido especialmente para Linux, o *JFS*, desenvolvido originalmente pela IBM mas também liberado com código aberto, e o mais conhecido de todos: o ext3, desenvolvido pelo Dr. Stephen Tweedie juntamente com outros colaboradores, na *Red Hat*[13].

O sistema de arquivos escolhido para utilização no *cluster* foi o *JFS*, pois além de possuir bom desempenho e baixo consumo de processador, é o mais rápido no processo de montagem, segundo testes[14] realizados.

4.2.2 Configuração de rede

As máquinas que irão compor o *cluster* possuem duas interfaces de rede cada. A primeira interface será conectada na rede 192.168.0.0/24 e é por onde a aplicação responderá. A segunda interface será dedicada para o *DRBD* fazer a replicação dos dados e será conectada ponto a ponto interligando os dois nós do *cluster* através de um cabo *crossover*. A tabela 4.8 demonstra a configuração de rede em cada um dos nós do *cluster*:

Tabela 4.8: Configuração de rede dos nós

HOSTNAME	IP ETH0	IP ETH1	GATEWAY
host1	192.168.0.50/24	10.0.0.1/24	192.168.0.1
host2	192.168.0.51/24	10.0.0.2/24	192.168.0.1

13 http://www.redhat.com
14 Os testes podem ser conferidos em http://www.debian-administration.org/articles/388

As Figuras 4.2.2.1 e 4.2.2.2 apresentam os endereços atribuídos às interfaces de rede e a tabela de roteamento do *host1* e do *host2*.

```
host1:~# ip addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.50/24 brd 192.168.0.255 scope global eth0
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever

host1:~# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:55 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global eth1
    inet6 fe80::5054:ff:fe12:3455/64 scope link
       valid_lft forever preferred_lft forever

host1:~# ip route show
10.0.0.0/24 dev eth1  proto kernel  scope link  src 10.0.0.1
192.168.0.0/24 dev eth0  proto kernel  scope link  src 192.168.0.50
default via 192.168.0.1 dev eth0
```

Figura 4.3: Interfaces de rede e tabela de roteamento do host1

```
host2:~# ip addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:58 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.51/24 brd 192.168.0.255 scope global eth0
    inet6 fe80::5054:ff:fe12:3458/64 scope link
       valid_lft forever preferred_lft forever

host2:~# ip addr show dev eth1
2: eth1: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:57 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 brd 10.0.0.255 scope global eth1
    inet6 fe80::5054:ff:fe12:3457/64 scope link
       valid_lft forever preferred_lft forever

host2:~# ip route show
10.0.0.0/24 dev eth1  proto kernel  scope link  src 10.0.0.2
192.168.0.0/24 dev eth0  proto kernel  scope link  src 192.168.0.51
default via 192.168.0.1 dev eth0
```

Figura 4.4: Interfaces de rede e tabela de roteamento do host2

É importante destacar que a aplicação responderá no endereço 192.168.0.53/24, que é o *IP* virtual do *cluster* e será configurado pelo *Heartbeat* na interface *eth0:0* do nó que estiver em estado primário. Os detalhes dessa configuração serão abordados na seção 4.2.5 desse capítulo.

4.2.3 Instalação e configuração do Exim

O *Exim* é o *MTA* padrão do *Debian* e portanto está incluso no pacote de instalação básico do sistema, dispensando a necessidade de instalação manual. Após a instalação do *Debian*, o *Exim* pode ser configurado utilizando o *Debconf*, uma ferramenta para gerenciamento da configuração do sistema. A Figura 4.5 demonstra o comando utilizado para configurar o pacote do *Exim*:

```
host1:~# dpkg-reconfigure exim4-config
```

Figura 4.5: Comando para configuração do Exim via Debconf
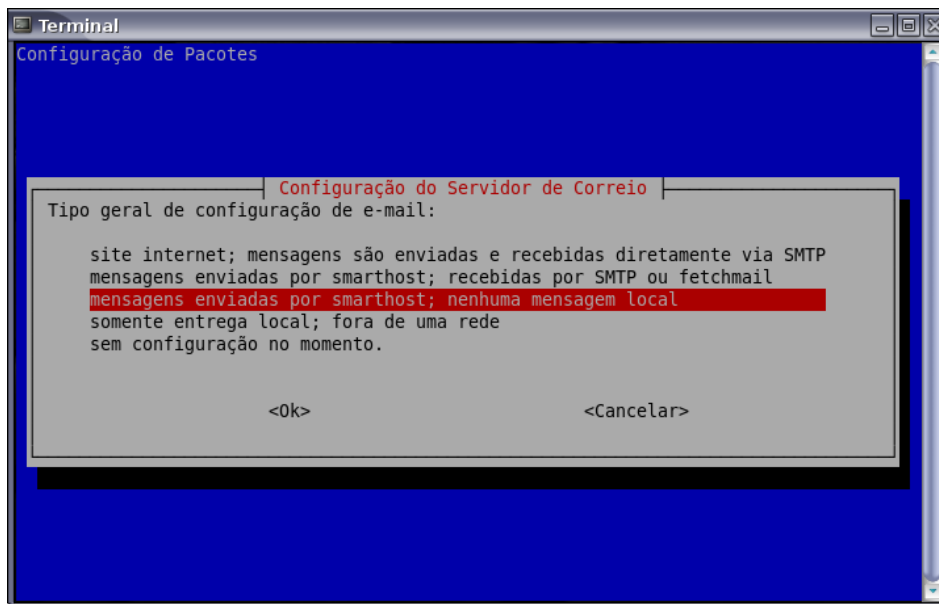
Figura 4.6: Configuração do Exim: Tipo de configuração

A primeira decisão que precisa ser tomada para configurar o *MTA* é o tipo de servidor que será instalado. O *Exim* oferece quatro opções, conforme demonstrado na Figura 4.6 e descrito a seguir:

- **Mensagens enviadas e recebidas diretamente via *SMTP***

Neste tipo de configuração é necessário um *IP* válido pois o roteamento de mensagens será feito diretamente via internet. Será preciso configurar os domínios para o qual o sistema receberá mensagens e o endereço das máquinas para o qual fará *relay.*

- **Mensagens enviadas por *smarthost;* recebidas por *SMTP* ou *fetchmail***

Neste tipo de configuração, todas as mensagens cujo destino seja um domínio diferente do que o *MTA* está respondendo, serão encaminhadas para outro servidor, denominado de *smarthost*, que por sua vez deve estar configurado para fazer o *relay* das mensagens.

- **Mensagens enviadas por *smarthost*; nenhuma mensagem local**

Este tipo de configuração é semelhante à anterior, no entanto, não haverá entrega de mensagens locais, ou seja, na própria máquina onde o *MTA* está sendo executado. Todas as mensagens serão encaminhadas para um *smarthost*.

- **Somente entrega local**

Neste caso deverá ser configurado os domínios para o qual o *MTA* deverá considerar a si mesmo como o destino final, portanto, todas as mensagens recebidas serão entregues localmente.

O *MTA* que será configurado para o *cluster* não estará conectado diretamente à internet e receberá somente mensagens oriundas da própria rede *192.168.0.0/24*. As mensagens cujo destino seja um domínio externo serão encaminhadas para um *smarthost*, portanto, a melhor opção neste caso é utilizar

a configuração **“Mensagens enviadas por *smarthost*; nenhuma mensagem local”**.

A próxima informação que será solicitada é o nome do sistema de correio, que deve ser o nome do domínio utilizado para qualificar as mensagens sem um nome de domínio. A Figura 4.7 ilustra o parâmetro utilizado neste caso.
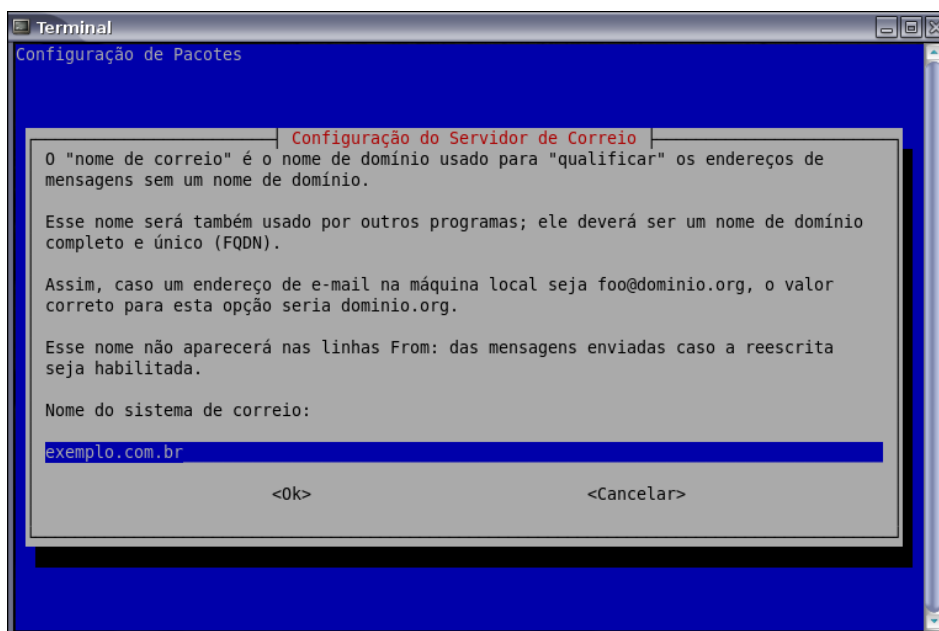
Figura 4.7: Configuração do Exim: Nome do sistema

Em seguida, é necessário informar as interfaces de rede por onde o *Exim* receberá requisições. Por padrão, está configurada a interface de *loopback*, no entanto, isso permitirá apenas o roteamento das mensagens enviadas pelos serviços locais. Para a configuração do *cluster*, o endereço *IP* virtual deve

adicionalmente ser informado. A Figura 4.8 ilustra essa configuração.
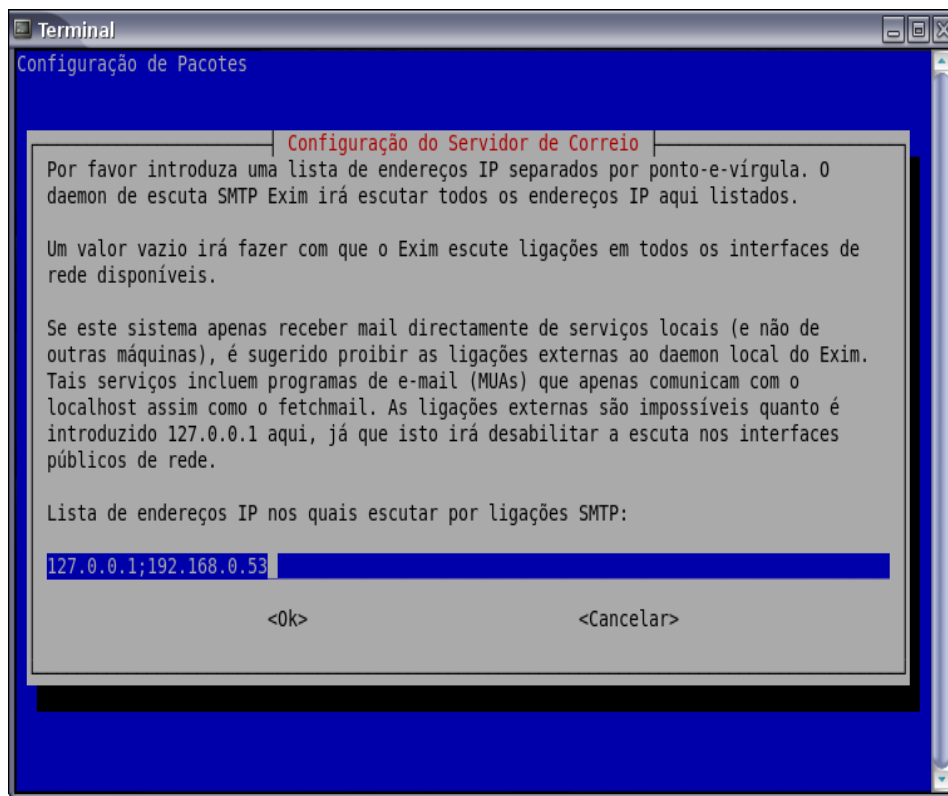
Figura 4.8: Configuração do Exim: IP por onde receber requisições

Por padrão, o *Exim* considera a si mesmo como destino final da mensagem o próprio *hostname* da máquina e o nome "**localhost**", isso significa, por exemplo, que mensagens destinadas a root@host1, root@localhost e root@host1.localhost serão sempre entregues localmente. No caso aqui apresentado, nenhum outro domínio deve ser informado, pois as mensagens que

não são locais devem ser encaminhadas para o *smarthost*.

O próximo parâmetro é o *hostname* ou endereço *IP* da máquina que será utilizada como *smarthost*, que deverá ser previamente configurada para fazer *relay* do *IP* virtual do *cluster*. No caso aqui apresentado será utilizado um provedor de serviços, responsável pelo roteamento das mensagens para a internet. A Figura 4.9 ilustra a configuração.
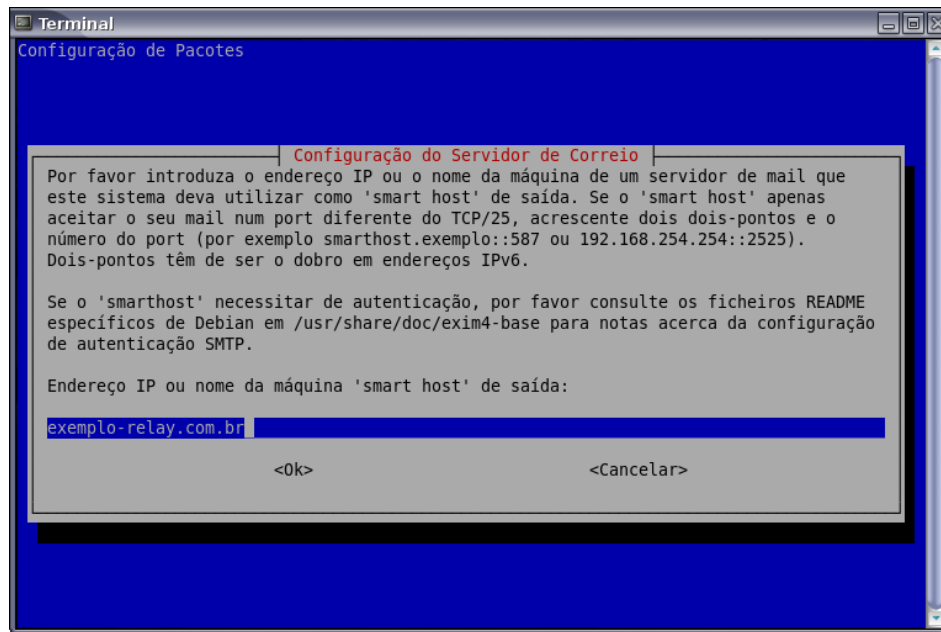
Figura 4.9: Configuração do Exim: Smarthost

Os parâmetros que foram apresentados aqui são essenciais para uma configuração básica do *Exim* e são suficientes para atender o objetivo proposto. O *Exim* oferece centenas de outros parâmetros que não estão disponíveis no

Debconf mas podem ser alterados diretamente nos arquivos de configuração, disponíveis no diretório */etc/exim4/conf.d*.

A inicialização e paralização do *Exim* será controlada pelo *Heartbeat*, abordado na Seção 4.2.5 deste capítulo, por isso, é necessário impedir a inicialização automática do serviço durante o *boot* do sistema, conforme demonstrado na Figura 4.10.

```
host1:~# update-rc.d -f exim4 remove
```

Figura 4.10: Impedindo a inicialização do Exim no boot do sistema

Para o correto funcionamento do *cluster*, as configurações apresentadas nesta seção devem ser feitas nos dois nós.

4.2.4 Instalação e configuração do DRBD

Até a data de conclusão deste trabalho, a versão do *DRBD* disponível no repositório do *Debian* “**estável**” é a 0.7.21. A instalação pode ser feita através de um pacote pré-compilado para a versão do *kernel* em uso, ou através do código fonte, utilizando o *module-assistant* do *Debian*.

Optou-se pela instalação através do código fonte com o objetivo de otimizar o *software* para a arquitetura utilizada. Isso pode trazer um aumento significativo de desempenho dependendo das características do *software* que será instalado.

Para utilizar esse método de instalação, é necessário que o código fonte do *kernel* do Linux e os arquivos de cabeçalho para a arquitetura utilizada estejam disponíveis no sistema. A Figura 4.11 demonstra como obter a versão do *kernel* em uso e o processo de instalação do código fonte e dos arquivos de cabeçalho do *Linux*:

```
host1:~# uname -a
        Linux host1 2.6.18-4-686 #1 SMP Mon Mar 26 17:17:36 UTC 2007 i686 GNU/Linux
host1:~# aptitude install linux-source-2.6.18
host1:~# aptitude install linux-headers-2.6.18-4-686
host1:~# tar jxf /usr/src/linux-source-2.6.18.tar.bz2 -C /usr/src
host1:~# ln -s /usr/src/linux-source-2.6.18 /usr/src/linux
```

Figura 4.11: Instalação do código fonte do kernel

Após o procedimento demonstrado na Figura 4.11 o ambiente está pronto para proceder com a compilação do módulo do *DRBD*. O *module-assistant* é uma ferramenta disponibilizada pelo *Debian* para facilitar a criação de módulos externos ao *kernel* do Linux e será utilizado para a compilação. Adicionalmente, é necessário instalar algumas ferramentas para controle e gerenciamento do *DRBD*, como o *drbdadm*, o *drbdsetup* e o *drbddisk*, disponibilizados pelo pacote *drbd0.7-utils*, e o *drbdlinks*, para gerenciamento de *links* simbólicos. A Figura 4.12 demonstra o procedimento de instalação dos pacotes assim como a compilação e instalação do módulo.

```
host1:~# aptitude install module-assistant
host1:~# aptitude install drbd0.7-module-source
host1:~# aptitude install drbd0.7-utils
host1:~# aptitude install drbdlinks
host1:~# m-a build drbd0.7-module
host1:~# dpkg -i /usr/src/drbd0.7-module-2.6.18-4-686_0.7.21-4+2.6.18.dfsg.1-12_i686.deb
```

Figura 4.12: Compilação e instalação do DRBD

O *DRBD* deve ser instalado nos dois nós do *cluster*, utilizando o mesmo procedimento. O *script* de inicialização */etc/init.d/drbd* é criado e configurado para execução automática durante o processo de *boot* da máquina e é responsável pelo carregamento do módulo *drbd.ko* no *kernel* do Linux.

Quando o módulo é carregado, o arquivo */proc/drbd* é criado e serve como interface para obter informações sobre o estado do sistema. A Figura 4.13 demonstra o conteúdo do */proc/drbd* obtido logo após a instalação.

```
host1:~# cat /proc/drbd
   version: 0.7.21 (api:79/proto:74)
   SVN Revision: 2326 build by root@host1, 2008-03-08 09:32:29
     0: cs:Unconfigured
     1: cs:Unconfigured
```

Figura 4.13: Conteúdo do /proc/drbd antes da configuração do sistema

O resultado obtido demonstra que o sistema ainda não foi configurado e por isso não está pronto para o uso. A configuração do *DRBD* pode ser feita através da ferramenta *drbdsetup* ou diretamente no arquivo */etc/drbd.conf*, que foi projetado para permitir o uso de arquivos idênticos em todos os nós do *cluster*, facilitando dessa forma o gerenciamento. O arquivo de configuração

consiste de seções e parâmetros. Cada seção é iniciada com um identificador, que em alguns casos pode ser acompanhado de um nome, seguido do caractere "{". O caractere "}" encerra a seção. A tabela 4.9 contém a relação de seções disponíveis e a tabela 4.10 os parâmetros.

Tabela 4.9: Seções do arquivo de configuração drbd.conf

SEÇÃO	DESCRIÇÃO
skip	Utilizada para comentar pedaços de texto. Tudo que estiver incluso em uma seção *skip* será ignorado.
global	Utilizada para a configuração de parâmetros globais. Na versão 0.7 do *DRBD* somente os parâmetros *minor-count*, *dialog-refresh* e *disable-ip-verification*, detalhados na tabela 4.10, são permitidos nessa seção.
resource *name*	Configura um recurso *DRBD*, ou seja, um dispositivo que será espelhado. Cada seção *resource* precisa conter obrigatoriamente duas seções *on host*, uma para cada nó do *cluster*, e adicionalmente uma seção *startup*, *syncer*, *net* e *disk*. O parâmetro *protocol* é obrigatório nesta seção e o parâmetro *incon-degr-cmd* opcional.
on *hostname*	Esta é uma seção que engloba os parâmetros para os dispositivos *DRBD* inclusos em uma seção *resource*. A palavra-chave *hostname* é mandatória e deve ser exatamente igual ao nome de *host* de um dos nós do *cluster*. Os parâmetros *device*, *disk*, *address* e *meta-disk* são obrigatórios nesta seção.
disk	Esta seção permite fazer configurações detalhadas (*tunning*) nas propriedades dos dispositivos de armazenamento do *DRBD*. Os parâmetros *on-io-error* e *size* são opcionais nessa seção.
net	Esta seção permite fazer configurações detalhadas (*tunning*) nas propriedades de rede do *DRBD*. Os parâmetros *sndbuf-size, timeout, connect-int, ping-int, max-buffers, max-epoch-size, ko-count* e *on-disconnect* são opcionais nesta seção.
startup	Esta seção permite fazer configurações detalhadas (*tunning*) nas propriedades de inicialização do *DRBD*. Os parâmetros *wfc-timeout* e *degr-wfc-timeout* são opcionais nessa seção.
syncer	Esta seção é utilizada para configurar em detalhes (*tunning*) o serviço de sincronização do dispositivo do *DRBD*. Os parâmetros *rate* e *group* são opcionais nessa seção.

Tabela 4.10: Parâmetros do arquivo de configuração drbd.conf

PARÂMETRO	DESCRIÇÃO
minor-count *valor*	O parâmetro *minor-count* deve ser utilizado quando é desejado adicionar *resources* no arquivo de configuração sem precisar recarregar o módulo do *DRBD*. O valor deve estar entre 1 e 255 e o padrão é exatamente o número de *resouces* configurados no arquivo *drbd.conf*.
dialog-refresh *tempo*	O *DRBD* exibe um texto que inclui um contador enquanto o nó aguarda pela conexão com o seu par. Esse parâmetro permite desabilitar essa funcionalidade ou aumentar o tempo de atualização do contador, o que pode ser interessante em casos onde a console do servidor está conectada a um terminal serial com capacidade limitada de *log*. O valor padrão desse parâmetro é 1, que corresponde a 1 segundo. O valor 0 desabilita completamente a atualização do contador.
disable-ip-verification	Este parâmetro proibe que o *DRBD* faça uso do comando *ip* ou *ifconfig* para verificações no endereço IP.
protocol *tipo*	Define o protocolo que será utilizado pelo *DRBD*. Os tipos disponíveis são: A, B e C, cujas características foram abordadas na Seção 4.1.3 do Capítulo 4.
incon-degr-cmd *comando*	Se um nó é iniciado em modo degradado, ou seja, como membro de um *cluster* que está operando com apenas um nó, e o *DRBD* identifica que esse nó possui inconsistência nos dados, o comando especificado nesse parâmetro será executado.
device *nome*	Define o nome do dispositivo de bloco que será utilizado por um *resource* do *DRBD*. Por exemplo: */dev/drbd0*.
disk *nome*	Define o nome do dispositivo de bloco que será utilizado por um *resource* para armazenamento de dados. Deve ser uma partição do disco dedicada para replicação do *DRBD*. Por exemplo: */dev/hda1*.
address *endereço:porta*	Cada *resource* do *DRBD* precisa de um endereço IP e uma porta TCP exclusiva para ser utilizada na comunicação com o seu par. Dois *resources* diferentes não devem utilizar a mesma porta TCP em um mesmo nó.
meta-disk *internal| dispositivo*	Define se a área de meta-dados será interna ou externa. Para definir uma área interna o valor *internal* deve ser utilizado e para definir uma área externa deve ser informado o dispositivo de bloco exclusivo que será utilizado. Por exemplo: */dev/hda6*.

on-io-error *ação*	Este parâmetro define a ação que será tomada pelo *DRBD* se o dispositivo de armazenamento reportar erro de *I/O*. As opções disponíveis são: ● ***pass_on***: Reporta o erro para o sistema de arquivos montado, se o erro acontecer no nó primário, e ignora o problema se o erro acontecer no nó secundário. ● ***panic***: O nó abandona o *cluster* reportando um *kernel panic.* ● ***detach***: O nó desconecta o dispositivo e continua a operação no modo *disk less*.
sndbuf-size *tamanho*	Define o tamanho do *buffer* de envio para o *socket* TCP. O padrão é 128K.
timeout *tempo*	Define o tempo que o nó aguardará por uma resposta esperada do seu par antes de considerá-lo indisponível. Este valor deve ser inferior aos valores dos parâmetros *connect-int* e *ping-int*, sendo o padrão igual a 60, que corresponde a 6 segundos.
connect-int *tempo*	Nos casos onde não é possível obter conexão com o dispositivo *DRBD* remoto, o sistema continuará tentando se conectar por um período de tempo indeterminado. Esse parâmetro define o intervalo de tempo entre duas tentativas e o valor padrão é 10, que corresponde a 10 segundos.
ping-int *tempo*	Se a conexão *TCP/IP* que estiver interligando um par de dispositivos *DRBD* estiver ociosa por um período de tempo maior do que o valor especificado nesse parâmetro, será gerado um pacote *keep-alive* para verificar se o par ainda está ativo. O valor padrão desse parâmetro é 10, que corresponde a 10 segundos.
max-buffers *número*	Define o número máximo de requisições que serão alocadas pelo *DRBD*. O valor mínimo é 32, que corresponde a 128KB em sistemas cujo *PAGE-SIZE* é igual a 4KB. Os *buffers* são utilizados para armazenar blocos de dados enquanto eles são gravados em disco, portanto, quanto maior for esse valor, maior será o desempenho do sistema e consequentemente o consumo de memória *RAM*.
ko-count *número*	Se o nó em estado secundário não completar uma requisição de escrita pelo tempo definido no parâmetro *timeout* multiplicado pelo valor desse parâmetro, então ele será expelido do *cluster* e o nó primário entrará no modo *StandAlone*. O valor padrão é 0, que desabilita essa funcionalidade.
on-disconnect *ação*	Este parâmetro define a ação que o *DRBD* deve tomar quando o nó perder a comunicação com o seu par. Os valores disponíveis são: *stand_alone*, para que o nó entre no modo *StandAlone* em situações como essa, ou *reconnect*, para que sejam feitas novas tentativas de conexão, sendo este o valor padrão do parâmetro.

on-io-error *ação*	Este parâmetro define a ação que será tomada pelo *DRBD* se o dispositivo de armazenamento reportar erro de *I/O*. As opções disponíveis são: ● ***pass_on***: Reporta o erro para o sistema de arquivos montado, se o erro acontecer no nó primário, e ignora o problema se o erro acontecer no nó secundário. ● ***panic***: O nó abandona o *cluster* reportando um *kernel panic.* ● ***detach***: O nó desconecta o dispositivo e continua a operação no modo *disk less*.
wfc-timeout *tempo*	O *script* de inicialização do *DRBD* interrompe o processo de *boot* da máquina e aguarda até que os *resources* estejam conectados. Esse comportamento evita que o gerenciador do *cluster*, no caso o *Heartbeat*, configure um dispositivo que "acha" ser o nó primário, enquanto não é. Se for desejado limitar o tempo de espera pela conexão com o par, é necessário informar o tempo em segundos nesse parâmetro. O valor padrão é 0 e significa que a espera pela conexão com o par é ilimitada
degr-wfc-timeout *tempo*	Esse parâmetro será utilizado apenas quando um nó que fazia parte de um *cluster* degradado, ou seja, um *cluster* que operava com apenas um nó, é reiniciado. Neste caso, o *degr-wfc-timeout* substitui o *wfc-timeout*, pois pode ser indesejável o nó aguardar pelo seu par que já estava inativo antes da reinicialização do *cluster*. O valor padrão desse parâmetro é 60, que corresponde a 60 segundos. Um valor igual a 0 faz com que a espera seja por um período ilimitado.
rate *velocidade*	Este parâmetro é utilizado para limitar a taxa de transferência entre os nós. O objetivo é garantir que a aplicação em execução no *cluster* não seja comprometida por um processo de sincronização total do *DRBD*. O valor padrão é 250, que corresponde a 250KBps. É permitida a utilização dos sufixos *K, M* e *G* para especificar o valor em *Kilobytes*, *Megabytes* ou *Gigabytes*.

Para a configuração do *cluster* foi criado um recurso chamado *exim*, utilizando o modo de operação síncrono através do *Protocolo C*, que garante um maior nível de confiabilidade para o processo de espelhamento dos dados. Foi configurado, através do parâmetro *incon-degr-cmd*, que o sistema deve ser desligado quando o nó iniciar como um *cluster* degradado e for identificado inconsistência nos dados deste nó.

Foram incluídas na configuração uma seção *startup*, definindo os parâmetros *wfc-timeout* e *degr-wfc-timeout* para 60 segundos e 120 segundos respectivamente, uma seção *disk* com o parâmetro *on-io-error detach*, definindo a ação que deve ser tomada pelo *DRBD* quando for encontrado erro de *I/O* no dispositivo de armazenamento, e uma seção *syncer* limitando em 10MB a taxa de transferência do processo de sincronização total.

As configurações do espelhamento foram feitas nas seções *on host1* e *on host2*. A definição da partição que será espelhada foi feita através dos parâmetros *device* e *disk* com os valores */dev/drbd0* e */dev/sdb1* respectivamente. Isso significa que fisicamente os dados serão armazenados na partição */dev/sdb1* mas, como essa partição está associada a um dispositivo *DRBD*, todo processo de leitura e escrita em disco deve ocorrer através do dispositivo */dev/drbd0.* Portanto, a partição onde estarão os dados da aplicação deve sempre ser montada utilizando o dispositivo de bloco */dev/drbd0* e nunca */dev/sda1*.

Nas seções *on host1* e *on host2* também foram configuradas as interfaces de rede exclusivas para a operação de espelhamento. No *host1* será utilizada a interface 10.0.0.1 e a porta 7788 e no *host2* a interface 10.0.0.2 e a porta 7788. Estas interfaces estão conectadas diretamente através de um cabo *crossover*, o que reduz a possibilidade de falhas, já que a comunicação não depende de outros dispositivos de rede como *hubs* ou *switches.*

Em ambos os nós a área de meta-dados do *DRBD* será interna, ou seja, na própria partição onde serão armazenados os dados da aplicação. O parâmetro *meta-disk internal* define essa configuração.

Essas foram as únicas configurações necessárias para o *DRBD*. Os demais parâmetros disponíveis foram omitidos e portanto o valor padrão de cada

um será utilizado, o que atende a necessidade do *cluster*. As mesmas configurações devem ser feitas nos dois nós, inclusive é recomendado que após a configuração de um nó, o arquivo *drbd.conf*, apresentado na Figura 4.15, seja copiado para o seu par.

Após a configuração, o serviço do *DRBD* pode ser reiniciado nos dois nós, que a princípio estarão ambos em estado secundário. Isso acontece porque o sistema ainda não tem condições de saber qual deles deve ser o primário. O comando *drdbsetup* será utilizado para eleger o *host1* como primário e dessa forma iniciar o espelhamento com o *host2*. A Figura 4.14 demonstra esse processo assim como o conteúdo do arquivo */proc/drbd* antes e após a eleição do *host1*.

```
host1:~# cat /proc/drbd
  version: 0.7.21 (api:79/proto:74)
  SVN Revision: 2326 build by root@host1, 2008-02-21 20:22:14
    0: cs:Connected st:Secondary/Secondary ld:Inconsistent
    ns:0 nr:0 dw:0 dr:0 al:0 bm:2 lo:0 pe:0 ua:0 ap:0

host1:~# drbdsetup /dev/drbd0 primary --do-what-I-say

host1:~# cat /proc/drbd
  version: 0.7.21 (api:79/proto:74)
  SVN Revision: 2326 build by root@host1, 2008-02-21 20:22:14
   0: cs:Connected st:Primary/Secondary ld:Consistent
   ns:0 nr:0 dw:0 dr:0 al:0 bm:2 lo:0 pe:0 ua:0 ap:0

host2:~# cat /proc/drbd
  version: 0.7.21 (api:79/proto:74)
  SVN Revision: 2326 build by root@host1, 2008-02-21 20:22:14
    0: cs:Connected st:Secondary/Primary ld:Consistent
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
```

Figura 4.14: Promovendo o host1 para primário

```
resource exim {

     protocol C;
       incon-degr-cmd "echo '!DRBD! pri on incon-degr' | wall ; sleep 60 ; halt
-f";

     startup {
          wfc-timeout 60;
          degr-wfc-timeout 120;
     }

     disk {
          on-io-error detach;
     }

     syncer {
          rate 10M;
     }

     on host1 {
          device     /dev/drbd0;
          disk       /dev/sdb1;
          address    10.0.0.1:7788;
          meta-disk  internal;
     }

     on host2 {
          device    /dev/drbd0;
          disk      /dev/sdb1;
          address   10.0.0.2:7788;
          meta-disk internal;
     }
}
```

Figura 4.15: Arquivo drbd.conf do cluster

O diretório */exim* foi criado manualmente para ser utilizado como ponto

de montagem da partição */dev/drbd0*, e é o local para onde a área de dados da aplicação deve apontar. A montagem do dispositivo será feita pelo *Heartbeat*, abordado na Seção 4.2.5 deste capítulo.

No *Debian*, o *Exim* mantém a sua configuração no diretório */etc/exim4* e no arquivo */etc/default/exim4.* O *spool* das mensagens é armazenado no diretório */var/spool/exim4*. É necessário que esses locais apontem para o diretório */exim*, que é o ponto de montagem do dispositivo *DRBD.* Dessa forma, os dados serão replicados entre os nós do *cluster* e estarão disponíveis quando o serviço for transferido para o nó secundário assim que o nó primário falhar.

O *DRBDLinks* é uma ferramenta utilizada para automatizar os apontamentos que precisam ser criados. Ele será utilizado pelo *Heartbeat* quando o nó de um *cluster* é alterado de secundário para primário. Os apontamentos devem ser configurados no arquivo */etc/drbdlinks.conf*, conforme demonstra a Figura 4.16.

```
mountpoint('/exim')
link('/etc/exim4')
link('/etc/default/exim4')
link('/var/spool/exim4')
```

Figura 4.16: Arquivo /etc/drbdlinks.conf

O parâmetro *mountpoint* deve conter o ponto de montagem utilizado pelo dispositivo *DRBD*, no caso */exim*. Os parâmetros *link* definem quais apontamentos serão criados quando o programa for executado. De acordo com a configuração apresentada na Figura 4.16, serão criados os seguintes links

simbólicos:

/etc/exim4 -> /exim/etc/exim4
/etc/default/exim4 -> /exim/default/exim4
/var/spool/exim4 -> /exim/var/spool/exim4

O *script /etc/ha.d/resource.d/drbdlinks*, criado durante a instalação do *DRBDLinks*, será utilizado pelo *Heartbeat* para criação dos apontamentos. Quando executado com o parâmetro *start*, o *script* renomeia os arquivos e diretórios existentes adicionando o texto "*.drbdlinks"* ao nome original e em seguida cria os *links* definidos no arquivo */etc/drbdlinks.conf*. Quando executado com o parâmetro *stop,* o *script* faz o inverso, ou seja, exclui os *links* simbólicos e renomeia os arquivos e diretórios para o seu nome original, sem o texto "*.drbdlinks"*. A configuração do *DRBDLinks* deve ser feita nos dois nós do *cluster*.

4.2.5 Instalação e configuração do Heartbeat

Até a data de conclusão deste trabalho, a versão do *Heartbeat* disponível no repositório do *Debian* "**estável**" é a 2.0.7. A Figura 4.17 demonstra o processo de instalação.

```
host1:~# aptitude install heartbeat-2
```

Figura 4.17: Instalação do Heartbeat

Durante a instalação do *Heartbeat*, o diretório */etc/ha.d* será criado e é o local onde os arquivos de configuração devem ser armazenados. O diretório */etc/ha.d/resources* armazena os *scripts* utilizados para o gerenciamento do *cluster*, como configuração de endereço IP, montagem de sistemas de arquivos, inicialização de serviços etc.

O *Heartbeat* mantém três arquivos de configuração, são eles: */etc/ha.d/ha.cf*, */etc/ha.d/haresources* e */etc/ha.d/authkeys*. Estes arquivos não são criados durante a instalação, portanto, precisam ser criados manualmente ou copiados do diretório */usr/share/doc/heartbeat-2*, onde encontram-se versões de exemplo e a documentação do *Heartbeat*.

O arquivo */etc/ha.d/ha.cf*, apresentado na Figura 4.18, é o principal arquivo de configuração. A tabela 4.11 contém a descrição de cada parâmetro utilizado.

```
debugfile        /var/log/ha-debug
logfile          /var/log/ha-log
keepalive        2
deadtime         5
warntime         2
initdead         60
serial           /dev/ttyS0
udpport          694
bcast            eth1
auto_failback    off
node             host1
node             host2
```

Figura 4.18: Conteúdo do arquivo /etc/ha.d/ha.cf

Tabela 4.11: Descrição das diretivas utilizadas na configuração do Heartbeat

PARÂMETRO	DESCRIÇÃO
debugfile	Especifica o arquivo de *log* do *Heartbeat* para mensagens de depuração.
logfile	Especifica o arquivo de *log* do *Heartbeat*.
keepalive	Especifica o intervalo de tempo, em segundos ou milissegundos se utilizado o sufixo ***ms***, entre os *heartbeats*.
deadtime	Especifica o intervalo de tempo, em segundos ou milissegundos, que o nó será declarado indisponível se não responder a um *heartbeat*. Este é um parâmetro de configuração muito importante e o valor utilizado deve ser analisado minuciosamente em cada caso. Um valor muito baixo pode causar uma falsa declaração de indisponibilidade do nó enquanto um valor muito alto pode gerar um retardo indesejável no processo de *failover*. O valor adequado pode variar dependendo da carga do sistema. Para chegar ao valor ideal desse parâmetro, é recomendado analisar o *log* por alguns dias e verificar se alertas "***heartbeat* atrasado**" aparecem. Se não aparecerem o valor escolhido deve ser adequado, caso contrário será necessário ajustar o valor.
warntime	Especifica quão rápido um alerta "***heartbeat* atrasado**" será emitido antes do nó ser considerado inativo. O valor recomendado deve estar entre ¼ e ½ o valor do *deadtime*.
initdead	Especifica após quanto tempo a partir da inicialização do *Heartbeat* um nó poderá ser declarado indisponível. Esse parâmetro é interessante em sistemas onde o serviço do *Heartbeat* é iniciado antes da comunicação entre os nós estar disponível. O valor recomendado é no mínimo duas vezes o valor do *deadtime*.
serial	Especifica a interface serial por onde os *heartbeats* serão enviados.
udpport	Especifica a porta *UDP* para ser utilizada na comunicação entre os nós.
bcast	Especifica a interface de rede por onde os *heartbeats* serão enviados.
auto_failback	Especifica se o processo de *failback* deve ser automático ou não.
node	Especifica os nós que fazem parte do *cluster*. O valor desse parâmetro deve corresponder ao *hostname* do nó.

O arquivo */etc/ha.d/haresources*, apresentado na Figura 4.19, deve ter versões idênticas em todos os nós e contém a configuração dos recursos que serão transferidos de um nó para o outro quando falhas são encontradas. O arquivo possui o seguinte formato:

[nome do nó] [recurso1] [recurso2] [recurso3] ... [recursoN]

O primeiro parâmetro não precisa necessariamente ser o nome do nó onde a configuração está sendo feita, mas sim, o nome do nó preferencial para executar a aplicação. Esta informação será utilizada pelo *Heartbeat* quando a diretiva *auto_failback on* for definida. Os recursos são *scripts* utilizados pelo *Heartbeat* quando são encontradas falhas em um nó do *cluster*.

```
host1 IPaddr::192.168.0.53/24/eth0 drbddisk::exim \
Filesystem::/dev/drbd0::/exim::jfs drbdlinks exim4
```

Figura 4.19: Conteúdo do arquivo /etc/ha.d/haresources

O *Heartbeat* busca os *scripts* no diretório */etc/ha.d/resources* e */etc/init.d*. Por padrão esses *scripts* devem aceitar como parâmetro os valores *start*, utilizado para iniciar recursos em um processo de *failover*, e *stop*, utilizado para finalizar recursos em um processo de *failback*. Outros parâmetros podem ser passados para os *scripts* utilizando o separador "***::***". A tabela 4.12 apresenta a descrição de cada recurso utilizado no arquivo */etc/ha.d/haresources*.

Tabela 4.12: Descrição dos recursos definidos no arquivo /etc/ha.d/haresources

RECURSO	DESCRIÇÃO
IPaddr	Este *script* é criado durante a instalação do *Heartbeat* e é utilizado para configurar o endereço IP virtual do *cluster*. Ele recebe como parâmetro o endereço IP, máscara de rede e a interface que será utilizada. No caso aqui apresentado, o endereço *192.168.0.53* com máscara de rede *255.255.255.0* será configurado na interface *eth0:0* quando o *script* for executado no modo *start*, ou seja, durante o processo de *failover*, e excluído quando o *script* é executado no modo *stop*, ou seja, em um processo de *failback*.

drbddisk	Este *script* é criado durante a instalação do *DRBD* e é utilizado para alterar o estado do dispositivo de bloco espelhado. É importante ressaltar que os dispositivos de bloco espelhados pelo *DRBD* só podem ser montados no nó que estiver em estado primário. No caso aqui apresentado o recurso *exim* do *DRBD* será definido como primário durante o processo de *failover* ou secundário durante o processo de *failback*.
Filesystem	Este *script* é criado durante a instalação do *Heartbeat* e é responsável pela montagem do dispositivo quer será utilizado para armazenamento dos dos dados da aplicação. Ele recebe como parâmetro o dispositivo de bloco, o ponto de montagem e o tipo de sistema de arquivos utilizado. O caso aqui apresentado tem o mesmo efeito do comando: *mount -t jfs /dev/drbd0 /exim*
drbdlinks	Este *script* é criado durante a instalação do *DRBDLinks* e será utilizado pelo *Heartbeat* para criação dos *links* simbólicos da aplicação, abordado na Seção 4.2.4 deste capítulo.
exim4	Este é o *script* padrão de inicialização do *Exim*, disponível no diretório */etc/init.d*. O *Heartbeat* executará esse *script* para iniciar o serviço do *Exim* durante o processo de *failover* ou parar o serviço em um processo de *failback*.

O arquivo */etc/ha.d/authkeys* contém as diretivas que serão utilizadas pelo *Heartbeat* para autenticação dos nós e por questão de segurança deve ser lido apenas pelo proprietário do arquivo. A tabela 4.13 apresenta a descrição de cada diretiva disponível e a Figura 4.20 o conteúdo do arquivo */etc/ha.d/authkeys* utilizado no *cluster*.

Tabela 4.13: Diretivas de autenticação do Heartbeat

DIRETIVA	DESCRIÇÃO
CRC	Oferece segurança apenas contra o corrompimento dos pacotes de dados. É o método que exige menor esforço computacional, no entanto, é recomendado apenas em ambientes seguros de rede.
SHA1	Utiliza criptografia com chave de 160 bits. Oferece maior segurança dentre as diretivas disponíveis e exige maior esforço computacional.
MD5	Utiliza criptografia com chave de 128 bits. Oferece nível médio de segurança dentre as diretivas disponíveis.

```
auth 1
1 crc
```

Figura 4.20: Conteúdo do arquivo /etc/ha.d/authkeys

5 VALIDAÇÃO DO *CLUSTER*

Após a instalação e configuração de todo o conjunto de *software* apresentado neste capítulo, o *cluster* está pronto para ser inicializado. A Figura 5.1 ilustra o funcionamento do *cluster* após inicialização dos dois nós. Inicialmente o *host1* assume a posição de primário, pois foi especificado como nó preferencial na configuração do *Heartbeat*. O endereço IP *192.168.0.53* é atribuído a interface *eth0:0* do *host1*, o dispositivo de bloco do *DRBD* é montado em */exim*, os links simbólicos que direcionam os diretórios de configuração e *spool* do *Exim* para a partição espelhada são criados e o serviço *SMTP* é inicializado.
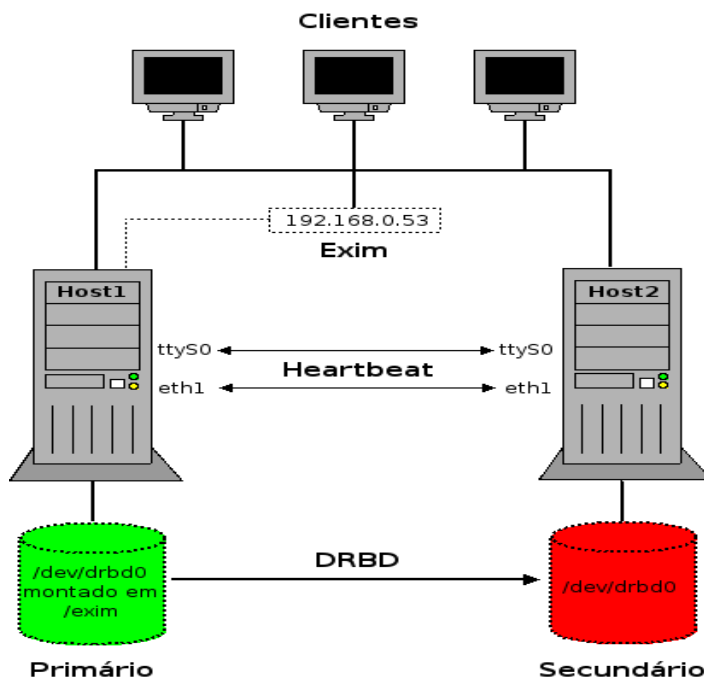
Figura 5.1: Cluster operando com o host1 no estado primário

A Figura 5.2 demonstra o conteúdo do arquivo */proc/drbd* nos dois nós, confirmando o estado primário do *host1*, o estado secundário do *host2* e a consistência dos dados replicados, conforme esperado.

```
host1:~# cat /proc/drbd
   version: 0.7.21 (api:79/proto:74)
   SVN Revision: 2326 build by root@host1, 2008-02-21 20:22:14
     0: cs:Connected st:Primary/Secondary ld:Consistent
     ns:56 nr:0 dw:12 dr:192 al:0 bm:2 lo:0 pe:0 ua:0 ap:0

host2:~# cat /proc/drbd
   version: 0.7.21 (api:79/proto:74)
   SVN Revision: 2326 build by root@host1, 2008-02-21 20:22:14
     0: cs:Connected st:Secondary/Primary ld:Consistent
     ns:0 nr:56 dw:56 dr:0 al:0 bm:2 lo:0 pe:0 ua:0 ap:0
```

Figura 5.2: Host1 primário - conteúdo do /proc/drbd nos dois nós

A Figura 5.3 demonstra o resultado da listagem das partições montadas em ambos os nós, comprovando que o dispositivo */dev/drbd0* foi montado corretamente no nó primário e está pronto para receber dados e replicá-los para o dispositivo remoto.

```
host1:~#  grep -i exim /proc/mounts
 /dev/drbd0 /exim jfs rw 0 0

host2:~#  grep -i exim /proc/mounts
```

Figura 5.3: Host1 primário - listagem das partições montadas nos dois nós

A Figura 5.4 demonstra que todos os *links* simbólicos que direcionam a configuração e o *spool* do *Exim* foram criados pelo *DRBDLinks* e que a aplicação está rodando somente no nó primário, conforme esperado.

```
host1:~# ls -l /etc/exim4 /etc/default/exim4 /var/spool/exim4
  lrwxrwxrwx 1 root root 23 2008-03-16 10:52 /etc/default/exim4 -> /exim/etc/default/exim4
  lrwxrwxrwx 1 root root 15 2008-03-16 10:52 /etc/exim4 -> /exim/etc/exim4
  lrwxrwxrwx 1 root root 21 2008-03-16 10:52 /var/spool/exim4 -> /exim/var/spool/exim4

host1:/etc/exim4# nc 192.168.0.53 25
  220 host1.drbd ESMTP Exim 4.63 Sun, 16 Mar 2008 11:08:57 -0300
```

Figura 5.4: Host1 primário - links simbólicos e execução da aplicação

Para efeito de teste, o *Exim* foi configurado para trabalhar no modo *queue_only*, ou seja, para não fazer o roteamento imediato das mensagens recebidas, deixando-as armazenadas no *spool*. Dessa forma, será possível comprovar a replicação dos dados entre os nós do *cluster*. A Figura 5.5 demonstra o recebimento de uma mensagem pelo *Exim* e em seguida a disponibilidade dessa mensagem no *spool*.

Após confirmar o correto funcionamento do *cluster* com os dois nós ativos e o *host1* no modo primário, é necessário confirmar se todos os recursos serão transferidos para o *host2* e o serviço *SMTP* estará disponível após falha do *host1*. Para simular essa situação, o *host1* foi desligado. A Figura 5.6 ilustra o *cluster* degradado após desligamento do *host1*.

```
host1:~# nc 192.168.0.53 25
    220 host1.drbd ESMTP Exim 4.63 Sun, 16 Mar 2008 11:46:12 -0300
    helo teste
    250 host1.drbd Hello root at teste [192.168.0.53]
    mail from:<teste@host1>
    250 OK
    rcpt to:<mickey@mouse>
    250 Accepted
    data
    354 Enter message, ending with "." on a line by itself
    Mensagem de teste
    .
    250 OK id=1Jau8T-00013h-VQ
    quit
    221 host1.drbd closing connection

host1:~# ls -l /var/spool/exim4/input/
    -rw-r----- 1 Debian-exim Debian-exim 861 2008-03-16 11:47 1Jau8T-00013h-VQ -D
    -rw-r----- 1 Debian-exim Debian-exim 677 2008-03-16 11:47 1Jau8T-00013h-VQ -H
```

Figura 5.5: Exim - recebimento de mensagem no modo queue_only
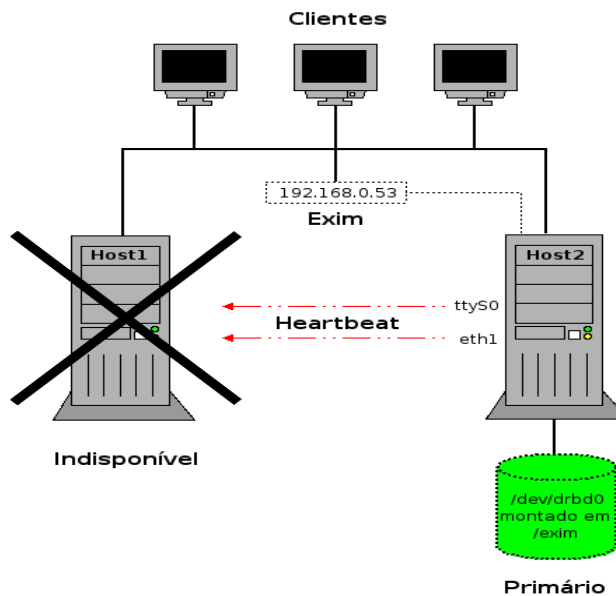
Figura 5.6: Cluster degradado

Através do arquivo de *log* do *Heartbeat*, exibido na Figura 5.7, é possível observar que a falha foi detectada e os recursos foram transferidos para o *host2*.

```
heartbeat[2116]:             2008/03/16_14:49:38 WARN: node host1: is dead
heartbeat[2116]:             2008/03/16_14:49:38 INFO: Cancelling pending standby operation
heartbeat[2116]:             2008/03/16_14:49:38 INFO: Dead node host1 gave up resources.
heartbeat[2116]:             2008/03/16_14:49:38 INFO: Link host1:/dev/ttyS0 dead.
heartbeat[2116]:             2008/03/16_14:49:38 INFO: Link host1:eth1 dead.
Filesystem[2727]:            2008/03/16_14:49:39 INFO: Running status for /dev/drbd0 on /exim
Filesystem[2727]:            2008/03/16_14:49:39 INFO: /exim is unmounted (stopped)
Filesystem[2663]:            2008/03/16_14:49:40 INFO: Filesystem Resource is stopped
ResourceManager[2300]:  2008/03/16_14:49:40 INFO: Running /etc/ha.d/resource.d/Filesystem /dev/drbd0
/exim jfs start
Filesystem[2836]:            2008/03/16_14:49:41 INFO: Running start for /dev/drbd0 on /exim
Filesystem[2772]:            2008/03/16_14:49:41 INFO: Filesystem Success
ResourceManager[2300]:  2008/03/16_14:49:42 INFO: Running /etc/ha.d/resource.d/drbdlinks  start
ResourceManager[2300]:  2008/03/16_14:49:42 INFO: Running /etc/init.d/exim4  start
heartbeat[2280]:             2008/03/16_14:49:56 INFO: all HA resource acquisition completed (standby).
harc[2972]:                  2008/03/16_14:49:56 INFO: Running /etc/ha.d/rc.d/status status
mach_down[2982]:             2008/03/16_14:49:56 INFO: Taking over resource group
                IPaddr::192.168.0.53/24/eth0
ResourceManager[3002]:  2008/03/16_14:49:57 INFO: Acquiring resource group: host1
                IPaddr::192.168.0.53/24/eth0 drbddisk::exim Filesystem::/dev/drbd0::/exim::jfs drbdlinks exim4
IPaddr[3026]:                2008/03/16_14:49:58 INFO: IPaddr Running OK
Filesystem[3211]:            2008/03/16_14:49:59 INFO: Running status for /dev/drbd0 on /exim
Filesystem[3211]:            2008/03/16_14:50:00 INFO: /exim is mounted (running)
Filesystem[3147]:            2008/03/16_14:50:00 INFO: Filesystem Running OK
mach_down[2982]:             2008/03/16_14:50:01 INFO: mach_down takeover complete for node host1.
heartbeat[2116]:             2008/03/16_14:50:01 INFO: mach_down takeover complete.
```

Figura 5.7: Arquivo /var/log/ha-log do host2

A Figura 5.8 confirma que após o desligamento do *host1*, todas as ações tomadas pelo *Heartbeat* no *host2* foram bem sucedidas e o serviço *SMTP* continua disponível.

Dando continuidade aos testes, o *host1* foi religado e observou-se que o *host2* manteve o estado primário no *cluster*, enquanto o *host1* assume o estado secundário. Este é o comportamento esperado, pois o *Heartbeat* foi configurado

para não executar o procedimento de *failback* automático. A Figura 5.9 ilustra o funcionamento do *cluster* com os dois nós ativos e o *host2* no estado primário.

O último teste realizado foi o desligamento do *host2*. Neste caso, observou-se o comportamento esperado, ou seja, o *Heartbeat* detectou a falha e transferiu todos os recursos para o *host1*, que estava operando no estado secundário até então e a partir desse momento tornou-se o nó primário no *cluster*.

```
host2:~# cat /proc/drbd
  version: 0.7.21 (api:79/proto:74)
  SVN Revision: 2326 build by root@host1, 2008-02-21 20:22:14
   0: cs:WFConnection st:Primary/Unknown ld:Consistent
   ns:0 nr:112 dw:124 dr:152 al:0 bm:2 lo:0 pe:0 ua:0 ap:0

host2:~# ip addr show eth0
  3: eth0: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
     link/ether 52:54:00:12:34:58 brd ff:ff:ff:ff:ff:ff
     inet 192.168.0.51/24 brd 192.168.0.255 scope global eth0
     inet 192.168.0.53/24 brd 192.168.0.255 scope global secondary eth0:0
     inet6 fe80::5054:ff:fe12:3458/64 scope link
       valid_lft forever preferred_lft forever

host2:~# grep -i exim /proc/mounts
  /dev/drbd0 /exim jfs rw 0 0

host2:~# ls -l /etc/exim4 /etc/default/exim4 /var/spool/exim4
  lrwxrwxrwx 1 root root 23 2008-03-16 14:49 /etc/default/exim4 -> /exim/etc/default/exim4
  lrwxrwxrwx 1 root root 15 2008-03-16 14:49 /etc/exim4 -> /exim/etc/exim4
  lrwxrwxrwx 1 root root 21 2008-03-16 14:49 /var/spool/exim4 -> /exim/var/spool/exim4

host2:~# nc 192.168.0.53 25
  220 host2.drbd ESMTP Exim 4.63 Sun, 16 Mar 2008 15:38:34 -0300

host2:~# ls -l /var/spool/exim4/input/
  total 8
  -rw-r----- 1 Debian-exim Debian-exim 861 2008-03-16 11:47 1Jau8T-00013h-VQ -D
  -rw-r----- 1 Debian-exim Debian-exim 677 2008-03-16 11:47 1Jau8T-00013h-VQ -H
```

Figura 5.8: Disponibilidade dos recursos no host2 durante indisponibilidade do host1
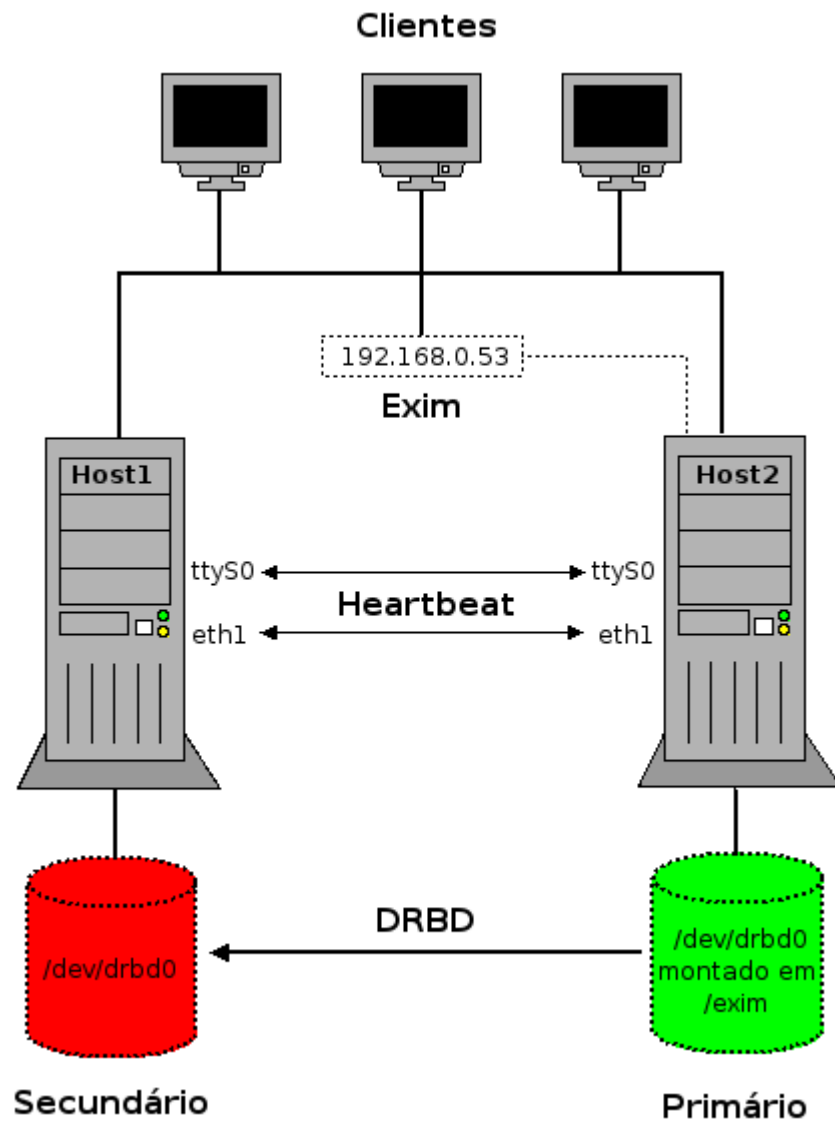
Figura 5.9: Cluster operando com o host2 no estado primário

6 CONCLUSÕES E TRABALHOS FUTUROS

Atualmente, a utilização de recursos tecnológicos possui notável relevância para os negócios de uma empresa e a conseqüência disso é uma dependência cada vez mais sólida da tecnologia e dos sistemas computacionais de um modo geral. Dessa forma, a interrupção não programada de um servidor pode resultar em prejuízos financeiros, perda de novos negócios ou até manchar a imagem da empresa, que por sua vez, precisa utilizar todos os artifícios disponíveis com o propósito de evitar esse acontecimento ou pelo menos contornar a situação a curto prazo.

A redundância de recursos é uma das maneiras mais adequadas para sanar o problema da indisponibilidade, no entanto, pode se tornar inviável devido ao custo elevado, como adquirir e manter uma solução proprietária de *cluster*, por exemplo.

Neste trabalho, foi realizado um estudo sobre as alternativas existentes para resolver o problema do fornecimento, com alta disponibilidade, do serviço *SMTP* em uma empresa. O requisito inicial foi a utilização de *software* livre. O maior desafio foi encontrar uma solução que além da redundância de *hardware* e *software*, também proporcione a redundância dos dados, pois, um sistema de armazenamento compartilhado tornaria-se um ponto único de falha, algo indesejado em uma solução de alta disponibilidade.

O objetivo foi alcançado com a utilização do sistema operacional GNU/Linux, o servidor *SMTP Exim* e o *software DRBD* e *Heartbeat*, para o espelhamento de dados em máquina remotas e monitoração dos nós do *cluster* respectivamente. Foram utilizados dois servidores com duas interfaces de rede

cada. A primeira interface é utilizada para as requisições ao serviço *SMTP*. A segunda interface interliga através de um cabo *crossover* os dois nós do *cluster* e é dedicada para a replicação dos dados e monitoramento dos nós. Após a instalação e configuração das ferramentas, cujo procedimento foi apresentado em detalhes no Capítulo 4, foram realizados testes para a validação da solução. Ao desligar o servidor que fornecia o serviço *SMTP*, simulando uma falha, o *Heartbeat* detectou com precisão o problema e executou todas as ações programadas transferindo o serviço para o servidor de contingência. Os dados espelhados pelo *DRBD* estavam atualizados e a falha simulada no teste foi prontamente contornada de forma transparente aos usuários do sistema, comprovando a eficácia da solução.

Como recomendação para trabalhos futuros, poderia ser implementado um *cluster* utilizando o *DRBD8* em conjunto com um sistema de arquivos para *cluster*, como o *OCFS2* ou o *GFS*. Uma solução deste tipo permite que o dispositivo de bloco espelhado seja montado simultaneamente nos dois nós, unindo as vantagens de um *cluster* de alta disponibilidade e de um *cluster* para balanceamento de carga em uma única solução.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ALVARENGA, F. V. U*ma Proposta de Aplicação para a Solução do Problema da Árvore Geradora de Custo Mínimo com Grupamentos Utilizando Cluster em Linux.* 2007. 63f. Monografia (Especialização em Administração em Redes Linux) – Departamento de Ciência da Computação, Universidade Federal de Lavras, Lavras.

DELL. *Understanding Storage Concepts* [on-line]. Disponível na Internet via www. url: http://support.dell.com. Arquivo capturado em 15 de janeiro de 2008

ELLENBERG, L. *Data Redundancy By DRBD*. 2003 [on-line]. Disponível na Internet via www. url: http://www.drbd.org/drbd-article.html. Arquivo capturado em 25 de fevereiro de 2008

ELLENBERG, L. *Shared-Disk semantics on a Shared-Nothing Cluster*. 2007 [on-line]. Disponível na Internet via www. url: http://www.drbd.org/fileadmin/drbd/publications/drbd8.linux-conf.eu.2007.pdf. Arquivo capturado em 23 de fevereiro de 2008

FERREIRA, R. E. *Linux: Guia do Administrador de Sistemas*. São Paulo: Novatec, 2003. 510p.

GARTNER, F. C. *Fundamentals of fault-tolerant distributed computing in asynchronous environments* [on-line]. Disponível na Internet via www. url: http://portal.acm.org. Arquivo capturado em 24 de junho de 2007

JÚNIOR, E. P. F.; FREITAS, R. B. *Construindo Supercomputadores com Linux: Cluster Beowulf*. 2005. 104f. Monografia (Tecnólogo em Redes de Comunicação) – Departamento de Telecomunicações, CEFET, Goiânia.

PEREIRA FILHO, N. A. S*erviços de Pertinência Para Clusters de Alta Disponibilidade.* 2004. 270f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

REISNER, P. *DRBD*. 2001 [on-line]. Disponível na Internet via www. url: http://www.drbd.org/fileadmin/drbd/publications/drbd_paper_for_NLUUG_2001.pdf Arquivo capturado em 23 de fevereiro de 2008

SILBERSCHATZ, A.; GALVIN, P. B.; CAGNE, G. *Sistemas Operacionais com Java*. Tradução da Sexta Edição. Editora Campus. 2004.

SIMÕES, E. I.; TORRES, G. M. *Alta Disponibilidade em Servidores*. 2003. 51f. Dissertação (Bacharel em Engenharia da Computação) – Escola de Engenharia Elétrica e de Computação, Universidade Federal de Goiás, Goiânia.

TAURION, C. [on-line]. Disponível na internet via www. url: http://www.timaster.com.br/revista/artigos/main_artigo.asp?codigo=530&pag=2. Arquivo capturado em 16 de maio de 2007.